DPU PROCESSING FOR XMM/OM

TRACKING AND COMPRESSION ALGORITHM

DOCUMENT 3

XMM-OM/PENN/TC/0004.02
Author: Cheng Ho
CH11/23/1990

Table of Contents

I. Overview

This document summarizes the current status of the algorithm design for the tracking and compression to be performed by the Digital Processing Unit of the Optical Monitor (OM) onboard the X-ray Multimirror Mission (XMM). This is intended to be a comprehensive summary which, standing on its own, covers the up-to-date development and thinking on various subjects. It should be keep in mind, however, that previous documentations and e-mails could contain discussions on various considerations at greater depth. It is assumed that the reader has at least a basic understanding how the Optical Monitor works, the role of the Digital Processing Unit and the operations and objectives of various observing modes. Baseline discussions are described in the January 89 proposal.

Following this overview, there are seven sections with supporting figures and tables. Section II discusses the analysis of stellar images and how we extract useful informations from the images. These informations are used as diagnostics for choosing good guide stars. Section III deals with the tracking algorithm based on a given number of guide stars. The calculated drifts are then fed to the compensation processing which corrects the drift and produces recovered images; the shifting-and-adding procedure is the subject of Section IV. Section V considers the compression of the image data via the Variable-Block-Tiered-Word-Length compression scheme. Section VI summarizes the general performance issues of the algorithm, such as the memory requirement and CPU cycle accounting. Section VII discusses the processing for the threshold mode and fast mode on the blue instrument. Section VIII gives a summary of "new" vocabularies used in this project.

Currently, the algorithm is developed and tested based on simulated data; we have not subject the algorithm to real data yet. The code was originally implemented in FORTRAN. However, the most current version is entirely coded in C. In the following, we describe the step-by-step operations in the simulation procedure. The steps marked with a (*DPU*) are those whose functions are expected to be performed in real-time by the DPU, i.e. these are the main products of this project. The others are for simulation only; in the real application, they will either be supplied by the detector or involves interface with external hardware components. When appropriate, I will indicate in which section detailed technical discussions can be found.

The following is an outline of the completed algorithm design and testing procedure in two major areas: 1) tracking and image accumulation and 2) compression of image data.

I.a. Data Generation and Tracking

(1) Generate a random field of stars, i.e. a catalog of stars is created using the star density distribution in Allen (p. 244). No spectral information is included.

(2) Generate a simulated image of 1024 by 1024 pixels with a given point spread function. The normalization of the image is such that a 15th mag star will

generate 600 counts per second in white light. The blue filter is implemented by multiplying the count rate by a factor of 0.133. This is the simulated *reference frame.*

(3) (*DPU*) Scan through the reference frame and pick out the bright stars above a certain threshold. Current implementation will pick out those star images which contain at least one pixel with more than n_{thld} counts where n_{thld} is a fixed value for the whole frame. This would typically pick out stars brighter than about 18th magnitude. The star image is "built" based on a summation around a local maximum. This yields a pair of numbers which is the star's coordinate, the counts of the star and the "size" of the image and other useful information which will help us select good guide stars. In the current implementation, only integer arithmetics is used. ≪ Section II≫

(4) (*DPU*) Scan through all the bright stars and pick out *good* guide stars. ≪Section II≫

(5) Begin subsequent frame simulation by first simulating the drift. The x and y translation from frame to frame is simulated by a random walk in random direction with a fixed step size: 0.3 arcsec=0.6 pixels. The roll from frame to frame is simulated as a linear (clockwise) rotation at 3.6 arcsec/frame; we simulate a linear roll to maximize the possible deterioration of the final image due to the roll. Within each frame, the roll angle is held fixed and the x and y translations are assumed to be linear. The pattern of drift and roll can be easily modified to simulate other situations.

(6) Modify the entire star catalog according to the drift. Use the new star catalog to simulate a frame. This is the working frame.

(7) (*DPU*) Analyze individual guide stars in the working frame. The algorithm is an abridged version of that for the construction of bright stars discussed in Step 3. The x and y location and the total count of the guide stars are obtained.

(8) (*DPU*) Calculate the drift and roll based on the locations of the guide stars in the current frame and those in the reference frame. The calculated drifts and roll will be downlinked in real time. ≪ Section III≫

(9) (*DPU*) Apply the calculated x and y drifts and performs a shift-and-add of the working frame to the 512 by 512 final image. ≪ Section IV≫

(10) Go to Step 5 and repeat 100 times.

(11) Write the 512 by 512 by 4-byte array into a binary file which is then processed and visualized using either IRAF or MONGO.

I.b. Data Compression

For the image data produced above, we have developed a reversible data compression scheme (the Variable-Block-Tired-Word-Length scheme). The compression algorithm is implemented and test in the following sequence.

(1) Read in the binary image file generated by Step 11 in the previous simulation.

(2) (*DPU*) Construct a raw data stream based on a 16 by 16 block in the image. The blocks in the entire image field are sequenced in a spiral fashion starting from the center of the image. ≪ Section V≫

(3) (*DPU*) Compress the raw data stream using the Variable-Block-Tiered-Word-Length scheme. ≪ Section V≫

(4) Write out the compressed data stream into a binary file.

To test the validity of the compression scheme, a decompress procedure is also developed to reverse the above process. The eventual mature descendant of the decompression routines is expected to be the front-end of the data reduction package on the ground.

II. Analyzing the Stellar Images

II.a. General Requirement

In order to perform the tracking, we need to know the locations of certain guide stars to some reasonable accuracy. A precision of a pixel (0.5 arcsec) is not sufficient. With a reasonable amount of on-board processing, we can easily do better than that. Furthermore, we need to be able to make judgement on the quality of the guide stars and select only the good ones. For example, single stars are good and binary images are bad. Images with overly extended wings or saturated pixels are bad. Stars too close to the boundary of the FOV are bad. To help make selection of good guide stars, we need further information which must be supplied by the on-board stellar analysis algorithm. Finally, we need to calculate the guide star locations for each frame and provide them to the frame-by-frame tracking algorithm.

II.b. Locating the Stars

We know where the stellar images are; they are made up of a number of pixels with counts above the background. The stellar image is spread out over a number of pixels due to the optics and the detector response. Given the blobs of bright pixels, we want to find out where the stars are. With a good knowledge of the point spread function and an infinite amount of processing power, the stellar location can be determined to very good accuracy. However, we do not have unlimited processing power on-board; we can only afford a simple and straightforward algorithm which does not involve fancy functions and square roots, etc.

The immediately intuitive approach is to take simple moments. The stellar locations are given by the centroids of the images which are obtained by calculating the first moments:

$$M_x = \sum_{ij} i \times N_{ij}, \qquad (II.1a)$$

$$M_y = \sum_{ij} j \times N_{ij}, \qquad (II.1b)$$

where i and j run along the x and y axis of the image field and N_{ij} is the number of photons in the (i,j) pixel. The location of the star is simply

$$x = M_x/C \qquad (II.2a)$$

$$y = M_y/C \qquad (II.2b)$$

where

$$C = \sum_{ij} N_{ij} \qquad (II.3)$$

is the total number of the photons in the region over which the summation is taken in equation (1). The questions are: *What determines the region of the summation? What is the manner of summation?*

Size of the field of summation The size of the field of summation cannot be too small. If it is smaller than, say, the FWHM of the PSF, then we are not sampling the images sufficiently. If the wings of the PSF, which carry great weight in helping us determine the stellar location, are not included, we will have great error in the calculated stellar location. On the other hand, the size of the field of summation cannot be too large. A large field of summation will encompass too many pixels whose photon counts are dominated by some finite background. Randomly fluctuating background will simply wash out any image feature we wish to study. Furthermore, a large field of summation could also include other stellar images which are practically independent of the one we wish to analyze. We are then not measuring the location of the star rather the center-of-mass of different stars. The optimal size of the field of summation is probably several times of the FWHM and the expected contribution from the wing of the PSF and the background are comparable to each other. Before we discuss in more details the size of the field of summation, let's consider the manner of summation over a given ixed region.

Manner of summation Suppose we have already determined the size of the field of summation for a single stellar image to be, say, 12 pixels. No doubt the summation region will be centered around the brightest pixel, the one with the largest number of counts. So, we can do a 12 by 12 summation in a double DO-loop (in a FORTRANistic way of saying things). At least two pitfalls occur in this approach. First, it is quite likely that the PSF will have some degree of axial symmetry. It is better that we sample the field in a similar fashion. A field of summation of 12 by 12 means that we are putting extra weight on the four corners which could mean an additional source of error. Second, restricting ourselves to a fixed region means that we cannot treat complex images. Even though we don't want to use complex images as guide stars, we still want to have a reasonable idea of the behavior of those images.

These two consideration implies that we want the size of the field of summation to be floating, which needs an algorithm which is *adaptive*, which means that we need to use *IF* statement (COMPARE-and-BRANCH) in the algorithm. To use these *IF* statements, we need to design criterions at which the summation terminates. The obvious starting point is the brightest pixel. Then we can integrate outward. To mimic the axial symmetry, we want to do the summation outward in the radial direction. Unfortunately, there is no detectors like that. The closest thing to a summation in the radial direction is probably a summation scheme following the four quadrants (Figure II.1). We describe the implementation of the summation scheme below.

In Figure II.1, the summation is done along the direction of the arrows and in the order marked with each arrow, starting from the brightest pixel in the neighborhood which is found *before* the actual summation begins. Figure II.1 shows the summation sequence from the first to the fourth quadrants. The particular order of the summation from quadrant to quadrant is not important. But following consecutive quadrants does save a little bit in overhead. The flowchart of the summation scheme is shown in Figure II.2. The range of integration is controlled by

the two numbers k and B. The first number k controls the termination level based on the already accumulated information on the stellar image. The number we use at this point is

$$k = 500. \qquad (II.4)$$

This is approximately equivalent to a termination at the 3σ level for a Gaussian PSF. The second number B controls the termination level at which the contribution from the stellar image is of the same order as the background. In principle, we can try to do a local evaluation of the background level. However, this probably requires a great deal of additional processing and the potential benefit is not all that obvious. Keeping B constant throughout the frame should be sufficient. We have set

$$B = 3b \qquad (II.5)$$

where b is the background counts per pixel in each frame. In the current implementation, the fixed background level is determined by taking the average of the pixels in a sample of three columns in the reference frame which are one-quarter, one-half and three-quarters of a frame from the frame boundaries (columns 265, 512 and 768 in the current definition of a frame). We note that B should always be nonzero, even in the case of narrow-band filter and very low background level. With the termination criterion, the summation routine keeps track of the bounds of the summation x_1, x_2, y_1 and y_2. These numbers will be used in the diagnostics discussed below.

The output number from the stellar-image routine is the total count C, the first moments M_x and M_y in the x and y directions, the bounds of the stellar image integration x_1, x_2, y_1 and y_2. In addition, we calculate the second moments of the image (not shown explicitly in the flowchart in Figure II.2), i.e.

$$M_{xx} = \sum_{ij} i^2 \times N_{ij} - x^2 C, \qquad (II.6a)$$

$$M_{yy} = \sum_{ij} j^2 \times N_{ij} - y^2 C, \qquad (II.6b)$$

$$M_{xy} = \sum_{ij} i \times j \times N_{ij} - xyC. \qquad (II.6c)$$

The numbers $x_1, x_2, y_1, y_2, M_{xx}, M_{yy}$, and M_{xy} will be used to diagnose the *goodness* of the stellar image. Before we discuss the diagnosing of the goodness of the stars, let's consider the precision of the centroiding routine.

II.c. Precision of the Stellar Locations

We now consider the precision from the centroiding routine. For simplicity, we shall consider a simple one-dimensional problem. Generalization to a two-dimensional image is straightforward. Suppose the 1-D PSF follows a accumulative probability

distribution function F characterized by a standard deviation of σ pixels in the digitized scheme. For definiteness, we shall adopt a Gaussian PSF unless otherwise noted. For a star located at x_0, the centroiding algorithm yields a calculated position whose average is

$$\bar{x} = \sum_i i \times \left[F\left(\frac{i + x_0 + 0.5}{\sigma}\right) - F\left(\frac{i + x_0 - 0.5}{\sigma}\right) \right]. \qquad (II.7)$$

The error of the calculated position is obtained by summing up the variance within each pixel

$$\delta\bar{x} = C^{-1/2} \sum_i \left[\left| (i - \bar{x})^2 \left[F\left(\frac{i + x_0 + 0.5}{\sigma}\right) - F\left(\frac{i + x_0 - 0.5}{\sigma}\right) \right] \right| + \left| (i - \bar{x})^2 b/C \right| \right], \qquad (II.8)$$

where C is the total number of counts in the stellar image and b is the background per pixel.

Several factors determine the accuracy of the centroiding procedure. The first factor is the sampling which is equivalent to $1/\sigma$. If the image is under-sampled (large σ), then we will suffered systematic error when the center of the star is not located near the the point of any symmetry, i.e., far from the edge or the center of a pixel. This occurs when $\sigma < 1$. For example, for the case of $\sigma = 0.5$, $\bar{x}$ deviates from x_0 by $\bar{x} - x_0 = 0.0023$ pixels even under the most ideal situation of $b = 0$ and summation all the way out to 10σ. This systematic error in principle can be calibrated out once we have a very good knowledge of the PSF. However, this is practically impossible for on-board processing. In general, as long as $\sigma \geq 1$, this systematic error is less than 0.001 pixel which is quite tolerable. The second factor is the size of the background b. The third factor is the range of summation, which in practice cannot be infinite. All these factors affect the systematic and random errors, $Dx = \bar{x} - x_0$ and $\delta\bar{x}$.

Without going into great details of the theoretical prediction of sampling and centroiding, we use the result for a well-sampled, zero-background and well-summed results of an image as a benchmark to judge the performance of the algorithm which we have implemented. Under the ideal situation of $\sigma \gg 1$, $b = 0$ and summation from $-\infty$ to $+\infty$,

$$\bar{x} = x_0 \qquad (II.9)$$

and the random error is characterized by a standard deviation of

$$\delta x = \sigma / C^{1/2}. \qquad (II.10)$$

Figure II.3 plots the deviation of the calculated star locations to the true stellar location as a function of the brightness of the stars. Figure II.4 plots the histogram of the normalized errors $\Delta x \equiv DxC^{1/2}/\sigma$ for the single stars. The stars plotted in figures II.3 and II.4 are selected from the *good* guide stars criterions discussed in the next section; i.e. we can be reasonably sure that we are studying images of single

stars. Also plotted in Figure II.4 is the expected distribution of a Gaussian noise with a standard deviation of unity. Keeping in mind the fact that the image field is simulated with a finite level of background and that the implemented algorithm carries the summation effectively out to the 3σ level, we conclude from Figure II.3 that the performance of the algorithm is satisfactory.

II.d. Integer Arithmetics with Expanded Precision

The discussions above show that the implemented algorithm does not generate outrageous error and we are relative close to an optimal algorithm. However, one additional issue in fact dictates the accuracy of the star locations in most cases. It has been emphasized by the hardware fabricators that floating-point arithmetics should be avoided whenever possible, even when it may mean a loss of accuracy in some calculations. The stellar location routine was specifically mentioned as a place where alternative to the floating-point arithmetics should be implemented. Current implementation calls for an all-integer arithmetics with an expanded precision by a factor of 8. When we calculate the moments, all numbers related to the position information are multiply by this expansion factor. (In the real implementation, they are actually left-shifted by three bits.) The calculated x and y locations include this expansion factor which is carried over into the tracking calculation discussed in Section III. The expansion factor is removed (by right-shifting) from the calculated drifts to give results in the right units before the drifts are applied to the shift-and-add routines. The drifts with the expanded precision will be supplied to the Instrument Control Unit to control the tertiary mirror. In this scheme, the location of any star is know to an accuracy of $\pm 0.5/8 = \pm 0.0625$ pixels.

In contrast, the guide stars in a simulated 10-s frame in white-light typically contain more than 2000 photons; the precision by the centroiding process alone is about $\sigma/(2000)^{1/2} \sim 0.022\sigma$. If we take 41 microns (convolving 36 microns of detector response with 20 microns from the optical system) to be the FWHM of the stellar image and 18 microns to be the size of a pixel and assuming the PSF is Gaussian, the standard deviation of the PSF is about $\sigma \simeq 1$ pixel. We see that the uncertainty caused by the usage of integer arithmetics actually outweighs that arises from photon statistics. Furthermore, the errorbar for different guide stars are now uniform, irrespective of the brightness of individual guide stars, as long as the guide star has more than about 300 counts (brighter than 18th magnitude). This uniformity of uncertainty helps us estimate the accuracy of the calculated drift. See the following section for more details.

In principle, we can further increase the expansion factor in the precision of the integers. However, in the implementation, it is found that going to an expansion factor of 16 or greater dramatically increases the complexity of the algorithm, especially in the calculation of the second moments which could cause overflow in a 4-byte integer representation. It does not appear that improvement in the accuracy warrants the additional effort and increased complexity in the algorithm.

II.e. Diagnostics for Good Guide Stars

The most important quantities from the stellar analysis routine are the locations of the stars. In addition, we get useful information which can help us make a judgement on the "qualities" of the bright stars for the purpose of tracking. What we want are single bright stars with well-defined image and little or no possibility of contamination and confusion. Even if the brightness of the star varies, we still can get an accurate position for the stars. From the star construction routine, we obtain the following numbers: (x, y) gives the centroid of the image; C gives the total count of photons in the image; (x_1, x_2) and (y_1, y_2) give the bounds of the summation of the stellar image; M_{xx}, M_{xy}, and M_{xy} are the second moments of the image in the xx, yy and xy orientation. The following criterions have been developed to diagnose bad qualities of a stellar image. For an image satisfying each criterion, a value of "goodness" (it should probably be termed as badness) is assigned. Based on the the goodness of each bright star, the algorithm then choose the suitable stars for tracking. The goodness information will also be transmitted to the ground in real time. Decision can be made on the ground to override the default guide stars selection. In the following, we discuss the various goodness criterions.

a) To avoid a slanted image, we compare the location of the star and the algebraic center of the image. For example, if $|x - (x_2 + x_1 + 1)/2|$ is large, then the image probably contains some asymmetry. We want to avoid this. Any star satisfying this criterion will be assigned a goodness of 1.

b) To pick out an elliptical image, we check the "eccentricity" of the image. Eccentricity is defined as the ratio of the *sum* of the image sizes in the x and y direction to the *difference*. If the eccentricity is greater than a given threshold, the star image gets a goodness of 2.

c) To guard against an extended image, any stellar image that is greater than a threshold in each direction will be given a goodness of 4. This threshold can be roughly guessed based on our knowledge of the instrument and how the stellar images are built. For example, currently we are simulating the PSF as a Gaussian with a FWHM of about 2 pixel and the standard deviation is about 1 pixel. When summing the stellar image, we are typically terminating the summation at a level at or below 3σ. Thus the stellar size criterion should be somewhat greater than 6 pixels. Current implementation uses 10 pixels.

d) To further guard against an extended image, we test the size of M_{xx} and M_{yy}. Stars with M_{xx} or M_{yy} above a threshold will be assigned a goodness of 8.

e) To further guard against an oblate image, we test the second eccentricity defined by the ratio of $M_{xx} + M_{yy}$ and $|M_{xx} - M_{yy}|$. Stars with large second eccentricity will have a goodness of 16.

f) To guard against an image extended in the diagonal direction, we check the value of $|M_{xy}|$. Stars with $|M_{xy}|$ above a threshold will be assigned a goodness of 32.

g) To guard against the possibility that the guide star might drift out of view in the 1000-s integration, any star which lies within a certain distance from the border will be given a goodness value of 64.

h) To avoid the case of clustering of guide stars in a small region, we check the relative locations between good guide stars. Each selected good guide star will be assigned a zone of exclusion. Any subsequent candidate guide stars falling in the previously established exclusion zones will be given a goodness of 128. Current implementation of the exclusion zone is ± 40 pixels in either the x or y direction.

i) To ensure high accuracy, we avoid using bright stars which may contain saturated pixels. Stars with a C greater than a threshold will be given a goodness value of 256.

The score of goodness is binary coded. The locations, total counts and the goodness scores of 15 good guide stars and 10 bad (goodness not equal to zero) guide stars will be downlinked. Operationally, a bad star is defined as one which has a non-zero value of a "binary AND" between its goodness score with 511. If we want to override the default selection, the downlinked goodness score can be used on the ground to help make selections of the guide stars. The diagnostic criterions and the current value of the diagnostic threshold are summarized in Table II.1.

The tests developed here are designed to give great versatility. Each of the disgnostics is controlled by a threshold; the criterions can be individually turned off by setting the diagnostic threshold to some extreme value. Alternatively, we can turn off a criterion by setting the particular bit which represents the result of the binary coding. For example, if we defined a bad guide star as one which has a non-zero value of binary AND between its goodness score and 510, then the first criterion is effectively turned off.

II.f. Guide Star Scanning Scheme

We summarize below in operational steps how the guide stars in the reference frame are obtained.

1. We scan through every pixels in the frame. A pixel with a count higher than n_{thld} will trigger the bright star image construction.

2. The bright star image construction follows the procedure described in II.b. After the whole frame is processed, we get a bright star catalog which contains n_{bs} entries of information.

3. We sort the n_{bs} bright stars by their brightness C.

4. We apply the disgnostics to the brightest stars. This application we stop after we obtain say 15 good guide stars or exhaust all bright stars.

5. The 15 good guide stars plus 10 brightest bad guide stars will be downlinked. In real-time operation, the ground operator will be given a fixed amount

of time, say five seconds, to override the default of using the 10 brightest good guide stars.

6. Once the reference frame information is established, the DPU signals the ICU for the initiation of observation.

II.g. Locating the Guide Stars for Tracking

Based the information obtained from the analysis of the reference frame, 10 (or some number of) guide stars are selected, either by the on-board software as a default or by ground control override. These 10 guide stars will be used throughout the 100-frame accumulation. Only under very dramatic situations will we change the number of guide stars. These situations include the disappearance of a guide star, a big jump in the brightness (we may even want to keep these) or the guide star drifting out of view (if this happens, we may need to modify disgnostic g). In these catastrophes, we will simply reduce the number of guide stars and reset the tracking numbers (see eqs. [III.9]); no new guide stars will be added. Under normal circumstances, we only need to calculate the locations of the guide stars in each frame and monitor the count in the guide stars. Information on the image quality will not be needed nor calculated. In the current implementation, an abridged version of the stellar image analysis routines is used which yields only (x, y) and C.

Figure Captions for Section II

Figure II.1 — Summation scheme of image analysis procedure. After the brightest pixel is located, the summation is carried out in the specified sequence. See Figure II.2 for the flowchart. Note that arrays are stored in a column-major fashion (y coordinate runs outside) in FORTRAN and in a row-major fashion (x coordinate runs outside) in C. The algorithm will be most efficient if it follows the array storage scheme. In Figure II.1, the summation is optimized for FORTRAN. A transposition of the summation sequence is required for C.

Figure II.2 — Flowchart of the stellar image analysis procedure. The numbers k and B are determined prior to the analysis and set as a constant for the entire frame.

Figure II.3 — Deviation of the calculated location of "good" guide stars from their real location as a function of their magnitude. The solid squares give the x deviation and the open triangles give the y deviation. The star field is simulated to be one in the galactic plane direction in white light and each star is assumed to make an image given by a radial Gaussian distribution with FWHM=1.17 arcsec.

Figure II.4 — The distribution of normalized deviation of the stellar image centroid, i.e. $\Delta x = Dx/C^{1/2}\sigma$. The thick solid line gives the distribution deviation in x location and the thick dashed line gives the deviations in the y location. For comparison, a normalized Gaussian is plotted as the hashed histogram. Factors

contributing to the widening of the distribution from a perfect gaussian include the finite background, the limited range of image summation, the drifting of the image within a frame. Note that in making this plot, we use floating point arithmetics for the location of each star. Effect of the integer arithmetics does not enter for these results.

Table II.1

IMAGE DIAGNOSTIC

Input numbers: $C,\ x,\ y,\ x1,\ x2,\ y1,\ y2,\ M_{xx},\ M_{yy},\ M_{xy}$.

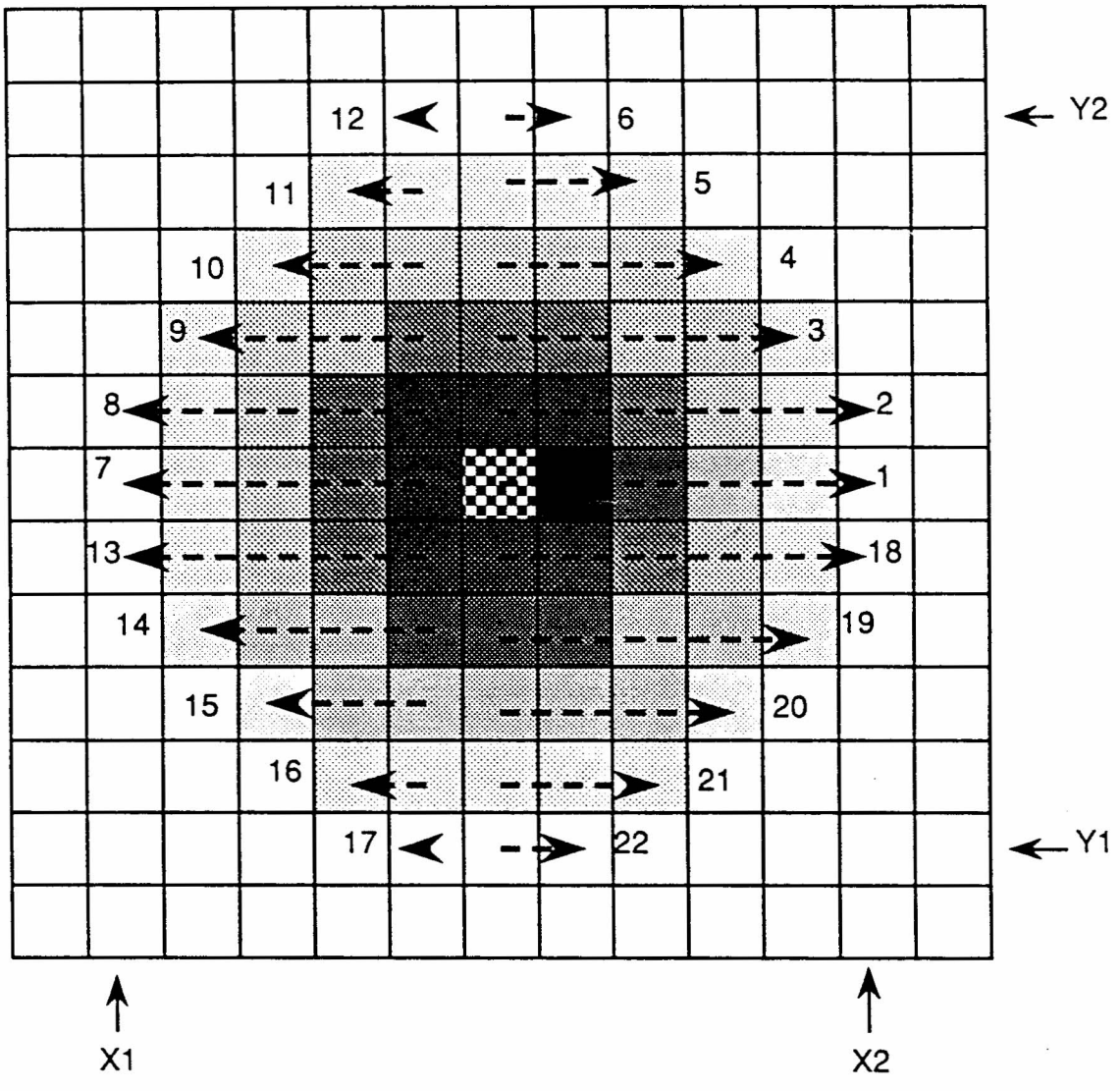
Diagnostic	Goodness
Off-Center Test, $x_c = (x2 + x1 + 1)/2, y_c = (y2 + y1 + 1)/2$ $\lvert x_c - x\rvert + \lvert y_c - y\rvert > d_1$	1
Oblateness Test, $D_x = x2 - x1, D_y = y2 - y1$ $D_x + D_y < d_2 \times \lvert D_x - D_y\rvert$	2
Size Test $D_x > d_3$ or $D_y > d_3$	4
Second Moment Test 1 $M_{xx} > d_4$ or $M_{yy} > d_4$	8
Second Moment Test 2 $M_{xx} + M_{yy} < d_5 \times \lvert M_{xx} - M_{yy}\rvert$	16
Second Moment Test 3 $\lvert M_{xy}\rvert > d_6$	32
Border Test $\lvert x - 512\rvert > 512 - d7$ or $\lvert y - 512\rvert > 512 - d7$	64
Saturation Test $C > C_{sat}$	128

Adjustable diagnostic parameters:

$d_1(6[E])$, $d_2(8)$, $d_3(10[P])$, $d_4(83[E])$, $d_5(7)$, $d_6(8[E])$, $d_7(450^2[P])$, $C_{sat}(20000)$.

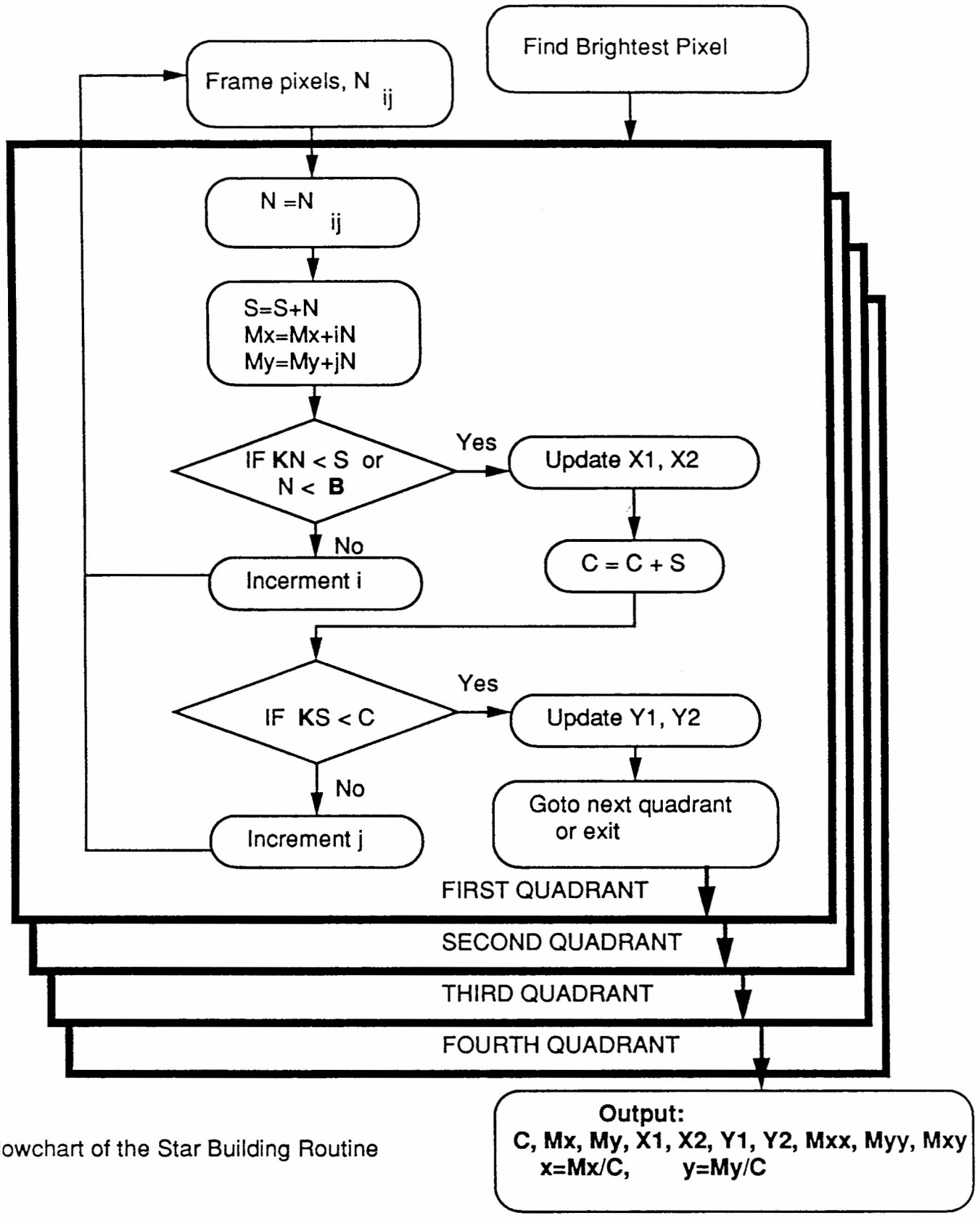
$[P]$: in units of pixels.

$[E]$: in units of integer representation of expanded precision ($[E] = [P]/8$).



Summation Scheme of the
Stellar Image Building Procedure

Figure II.1



Flowchart of the Star Building Routine

Figure II.2

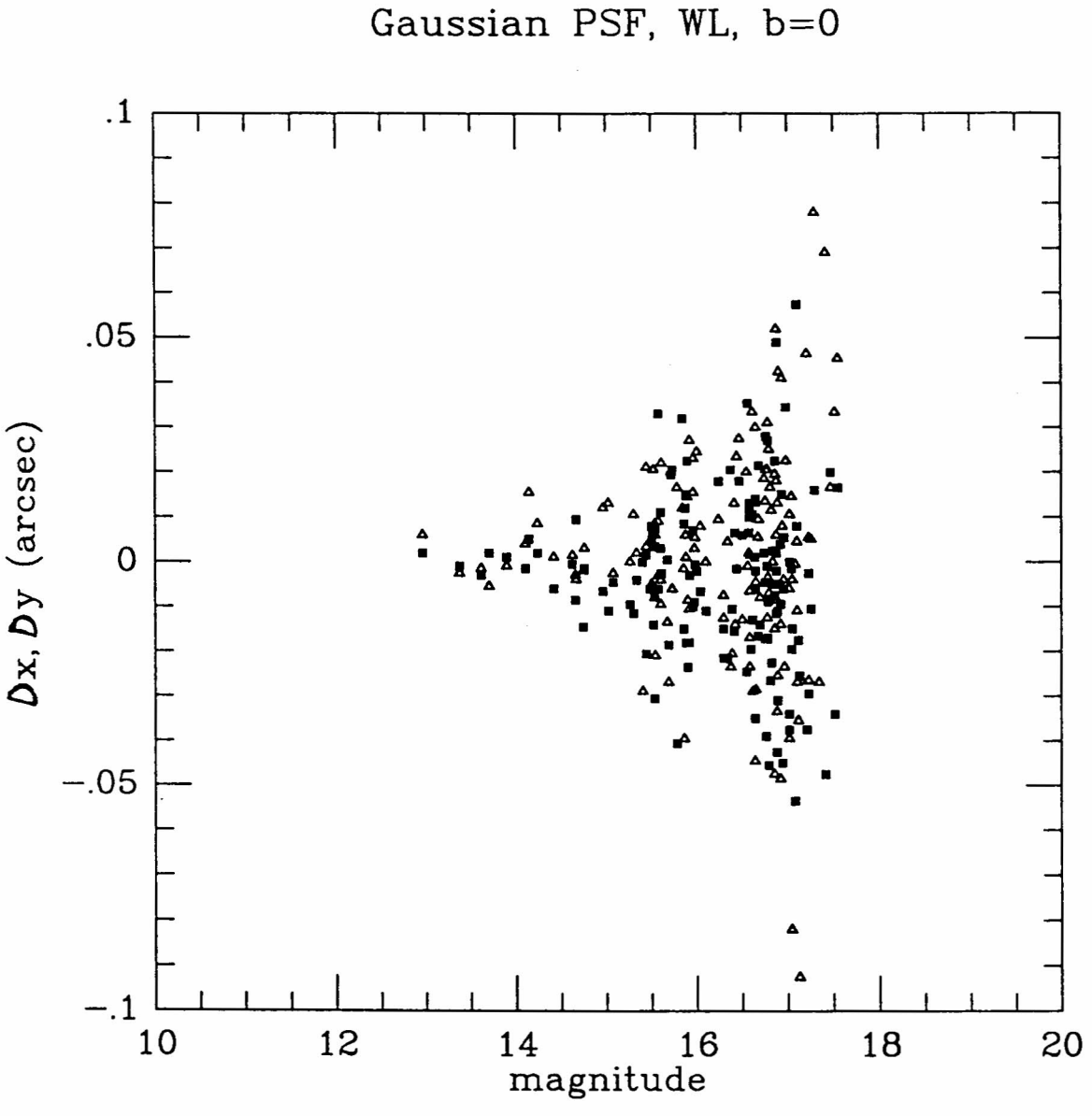

Gaussian PSF, WL, b=0
.1
.05
0
−.05
−.1
Dx, Dy (arcsec)
10
12
14
16
18
20
magnitude

Figure II.3

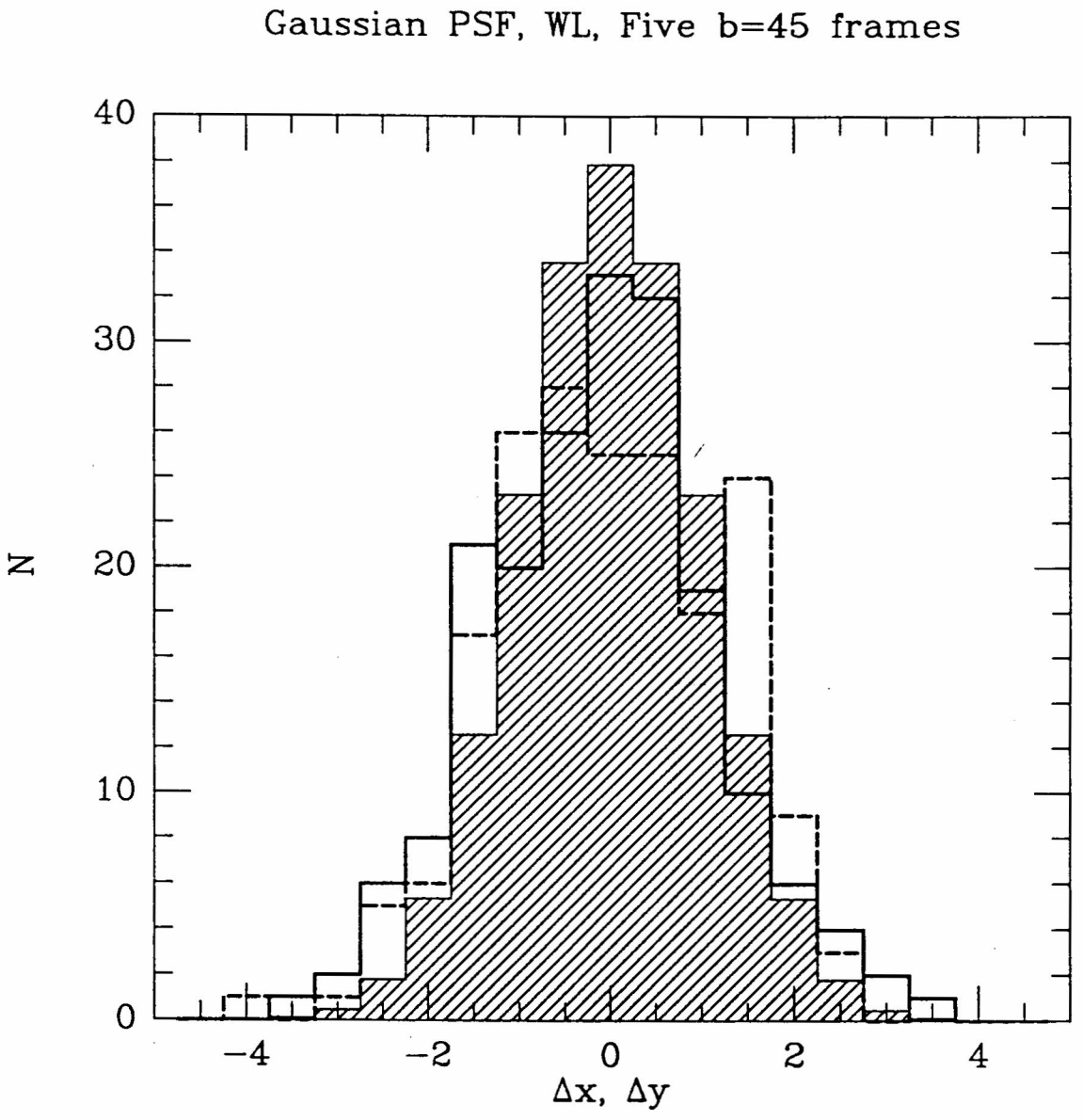


Figure II.4

III. Tracking

III.a. Basic Formalism

Suppose we have located and chosen N good guide stars within the field of view in the reference frame at (u_i, v_i) where i runs from 1 to N. At a later moment, we take an integration of a frame and determine the new locations of the guide stars at (x_i, y_i). Our task is to determine the change in the aspect of the current frame with respect to the reference frame, i.e. to calculate the drift in the x and y direction as well as the roll with respect to a particular roll center taken to be the origin. The selection of the roll center is *arbitrary*. However, see discussions below on a natural default choice of the roll center where the errors of the calculated drifts are *formally* minimized.

Applying a drift and roll $(\Delta x, \Delta y, \Theta)$ to the guide stars, their new locations are given by

$$\hat{x}_i = u_i C - v_i S + \Delta x \qquad (III.1a)$$

$$\hat{y}_i = u_i S + v_i C + \Delta y, \qquad (III.1b)$$

where $C = \cos\Theta$ and $S = \sin\Theta$. (Without causing any confusion, we use C here for the cosine of the roll angle, while in the rest of this document, C refers to the total count in a stellar image. In III.b., we shall see that the cosine equals to unity for all practical purposes.) With the observed (x_i, y_i), the drifts are calculated by minimizing

$$D^2 \equiv \sum_i \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]. \qquad (III.2)$$

Differentiating D^2 with respect to Δx, Δy and Θ yields

$$\Delta x = (X - UC + VS)/N \qquad (III.3a)$$

$$\Delta y = (Y - VC - US)/N \qquad (III.3b)$$

$$SP + CQ = 0, \qquad (III.3c)$$

where

$$U = \sum_i u_i, \qquad (III.4a)$$

$$V = \sum_i v_i, \qquad (III.4b)$$

$$X = \sum_i x_i, \qquad (III.4c)$$

$$Y = \sum_i y_i, \qquad (III.4c)$$

$$P = \left[\sum_i (x_i u_i + y_i v_i)\right] - (XU + YV)/N = (2N)^{-1} \sum_{ij} \left[(x_{ii} u_{ij} + y_{ij} v_{ij}\right], \quad (III.4d)$$

$$Q = \left[\sum_i (x_i v_i - y_i u_i)\right] - (XV - YU)/N = (2N)^{-1} \sum_{ij} \left[(x_{ij} v_{ij} - y_{ij} u_{ij}\right], \quad (III.4e)$$

and

$$x_{ij} = x_i - x_j, \quad y_{ij} = y_i - y_j, \quad u_{ij} = u_i - u_j, \quad v_{ij} = v_i - v_j.$$

For our application, the roll angle is always small and P is always greater than zero, thus the roll angle is given by

$$\Theta = \sin^{-1}\left[-Q/(P^2 + Q^2)^{1/2}\right]. \quad (III.5)$$

The tracking formalism can be generalized to incorporate the statistical weight of different guide stars by modifying equation (2) to

$$D'^2 \equiv \sum_i \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right] / \sigma_i^2. \quad (III.6)$$

where $\sigma_i = (C_i)^{1/2}$ is the standard deviation in the location and C_i the total count of the guide star i. However, this generalization introduces great complexities into the algorithm. Usage of equation (2) in place of (6) is justified for two reasons and demanded by a third reason. First, the algorithm will always use the brightest N *good* stars within the field of view for tracking. Thus, they would probably have similar brightness and statistical significance. In the simulation of an average star distribution in white-light, the brightest guide star is typically 5 times brighter than the least bright guide star, i.e. there is a factor of 5 difference in the statistical significance among guide stars. Keeping in mind that the precision of the tracking is only as good as the worst guide star, we will always obtain conservative estimate of the precision of the drift tracking. Second, the star locations will have an additional source of uncertainty. To calculate the guide star locations efficiently, we use integer arithmetics with the precision enhanced by a factor of 8, in place of floating-point arithmetics. (See II.d. for more detailed discussions.) This means that the calculated (x_i, y_i) will have a minimum errorbar of $\pm$ 0.0625 pixel= $\pm$0.0325 arcsec. This errorbar is generally greater than that from the photon statistics alone. As a result, the guide star locations will have a roughly uniform errorbar of $\pm$0.0325 arcsec. Hence, the usage of equation (III.2) is justified. Third, from a practical standpoint, the additional processing needed to utilize equation (III.6) will be prohibitively inefficient as far as on-board processing is concerned. In fact, we will implement additional simplifications in the small roll angle limit for the on-board processing as discussed below.

III.b. Small Roll Angle Approximation

As long as Θ is small, we can make the following approximation in calculating the roll angle,

$$C = 1, \qquad \text{and} \qquad S = \Theta = -Q/R, \tag{III.7}$$

where

$$R = \left[\sum_i (u_i^2 + v_i^2)\right] - (U^2 + V^2)/N = (2N)^{-1} \sum_{ij} \left[u_{ij}^2 + v_{ij}^2\right] \equiv Nr^2/2 \tag{III.8}$$

and r is the *average* distance between the guide stars. (In the limit of large N randomly distributed guide stars, $r \rightarrow d/\sqrt{3}$ where d is the linear dimension of the square FOV.) In the implementation, R as well as U and V need only be calculated once for the reference frame at the beginning of the tracking. Furthermore, calculating Θ via equation (III.7) does not need the square root operation as in equation (III.5). This approximation will introduce error which is of order Θ^2. Since the roll angle is expected to be on the order of 0.002 radians from the beginning to the end of a 1000-s integration, using equation (III.8) will not introduce any significant error (see below for estimate in the error of Θ). Note that the values of P, Q, R, r and Θ are independent of the location of the roll center.

In summary, the drift and the roll of the satellite are calculated through the following equations.

Universal Quantities

$$U = \sum_i u_i, \tag{III.9a}$$

$$V = \sum_i v_i, \tag{III.9b}$$

$$R = \left[\sum_i (u_i^2 + v_i^2)\right] - (U^2 + V^2)/N. \tag{III.9c}$$

Frame-Dependent Tracking Quantities

$$X = \sum_i x_i, \tag{III.10a}$$

$$Y = \sum_i y_i, \tag{III.10b}$$

$$Q = \left[\sum_i (x_i v_i - y_i u_i)\right] - (XV - YU)/N. \tag{III.10c}$$

Drift and Roll

$$S = \Theta = -Q/R, \tag{III.11a}$$

$$\Delta x = (X - U + VS)/N, \tag{III.11b}$$

$$\Delta y = (Y - V - US)/N. \tag{III.11c}$$

III.c. Precision of the Tracking

Since the observed location of the guide stars (x_i, y_i) as well as the reference locations (u_i, v_i) will no doubt have uncertainties, we expect the calculated drifts and roll to have some uncertainty. We now estimate the errorbar of the tracking quantities and the calculated drift and roll. For simplicity, we shall assume that the error bar for the stellar location is the same in the x and y direction and each stellar location is determined with the same standard deviation δx. The standard deviations for X, Y, U, and V are straightforward:

$$\delta X = \delta Y = \delta U = \delta V = \sqrt{N}\delta x. \tag{III.12}$$

Even though the values of $\delta X(\delta Y)$ and $\delta U(\delta V)$ are the same, we need to make a distinction between them. The quantities U and V are universal for the frame-by-frame tracking; errors associated with a linear dependence of them can be easily removed as a systematic error by an after-the-fact calibration to align the star positions with some absolute reference frame; the linear contributions of U and V in equations (III.11b) and (III.11c) does not affect the precision of the tracking.

The error of Q is

$$\delta Q = \sqrt{R}\delta x, \tag{III.13}$$

and the error of Θ is

$$\delta\Theta = \frac{\delta Q}{R} = \frac{\delta x}{\sqrt{R}}. \tag{III.14}$$

The standard deviations for the drifts are

$$\delta(\Delta x) = \frac{\delta x}{\sqrt{N}}\left[1 + \frac{V^2}{NR}\right]^{1/2}, \tag{III.15a}$$

and

$$\delta(\Delta y) = \frac{\delta x}{\sqrt{N}}\left[1 + \frac{U^2}{NR}\right]^{1/2}. \tag{III.15b}$$

As long as the roll center is chosen to lie within the field of view, both V^2/NR and U^2/NR will be less than unity.

Mathematically, applying a rotation and translation parameterized by the calculated drift and roll $(\Delta x \pm \delta(\Delta x), \Delta y \pm \delta(\Delta y), \Theta \pm \delta\Theta)$ to any particular point (x, y) point will yield the same result $(x' \pm \delta x', y' \pm y')$, *independent to the choice of the origin (roll center).* However, for efficiency reason, we will only perform on-board

compensation to the translation (shift-and-add) to the satellite drift; no compensation will be applied for the roll. The shift-and-add procedure leads to the smallest errorbar at the roll center and the error is greater for points at greater distances from the roll center. Figure III.1 shows the deviation of the calculated drift from the real (simulated) drift. Figure III.2 shows deviation of the calculated roll from the real (simulated) roll. The spread of the deviations is consistent with the estimates given in equations (III.14) and (III.15).

We now discuss a special case where the errorbars of Δx and Δy are *formally* minimized. This is achieved by minimizing U^2 and V^2, i.e., choosing the center-of-mass of the guide stars as the roll center and setting $U = 0$ and $V = 0$. This simplifies equations (11*b*) and (11*c*) to $\Delta x = X/N$ and $\Delta y = Y/N$. The center-of-mass is the natural default roll center. However, with a moderate increase in arithmetics, we shall implementing the tracking algorithm as equation (11) which does leave open the possibility of choosing the roll center at *any* location.

III.d. Final Image Quality

After the drifts in x and y are obtained, the algorithm will apply the compensation to the current frame by shifting-and-adding to the final 512 by 512 image. The operations of shift-and-add are mundane yet numerous. This has been recognized as a major task for the DPU onboard processing. The shift-and-add algorithm will be discussed in the next section. Here we discuss the quality of the final image as a result of the drifting and compensation. Figure III.3 shows three images of a 16-th magnitude star in white light after a 1000-s integration. Judging from the figures on the left and right, the stellar image quality is well retained by the process. This provide confidence to both the drift calculation and the shift-and-add procedure. Figure III.4 shows the accuracy of the stellar locations (with respect to the real position) by using a simple centroiding algorithm similar to that discussed in Section II. Other than the constant offset which can be calibrated out, we obtain better accuracy for the stellar location as a result of the better photon statistics. It is quite likely that we will be able to obtain astrometric accuracy better than 0.01 arcsec for stars brighter than 15-th magnitude.

Figure Captions for Section III

Figure III.1 –a– Deviation of the calculated drift from the real (simulated) drift in an average star field in white light. The solid line gives the result for the x drift and the dashed line for the y drift. Both are consistent with having an errobar of ± 0.01 arcsec $\sim \pm 0.03125/\sqrt{10}$ arcsec. The drift calculation utilizes the expanded precision arithmetics. Note the finite offset of the drifts from zero. This is the result of the finite error in the universal quantities calculated from the reference frame. This finite offset will be calibrated out in the image post-processing.

Figure III.2 — Deviation of the calculated roll from the real (simulated) roll. Other than the constant offset, we estimate the deviation to be within the range of $\pm 6 \times 10^{-5}$ radians. In the simulated field, this is consistent with the estimate obtained from Equation (III.14). (In the limit of a large number of guide stars, we estimate the error in Θ by taking $r \sim 448/\sqrt{3} \simeq 259$ arcsec. Here the size of the FOV includes the finite width of the border which we impose on the good guide star criterion. Combining equations [III.8] and [III.14] and adopting δx=0.03125 arcsec, we get $\delta\Theta \simeq 5.4 \times 10^{-6}$ radians.)

Figure III.3 – Performance of the tracking and shift-and-add procedure. The left figure shows the result of the star imaged by a perfectly stable satellite. The contour levels are 316, 1000, 3160, 10000, and 31600 from outside in. The center figure shows the stellar image after a simulated random-walk drift of the satellite. No compensation is applied to the frame summation. The trajectory of the satellite drift is shown as the solid boxes. The right figure shows the image taken by a drifting satellite (following the drift trajectory in the center figure) and compensated according to the result of the tracking calculation. Without going into detailed image analysis, the tracking and shift-and-add appears to retain most of the image quality.

Figure III.4 – Deviations of calculated stellar locations from their real locations as a function of star brightness. The solid squares give the x deviaiton and open triangles gives the y deviations. The stellar locations are calculated with a centroiding algorithm similar to that used in Section II. The sample of stars are those picked out by the good guide star diagnostics of Section II. Closer examination shows that some of the outlier stars are contaminated by some neighboring faint stars. Astrometric performance can in fact exceed that shown in the figure, once a more thorough analysis method is applied.

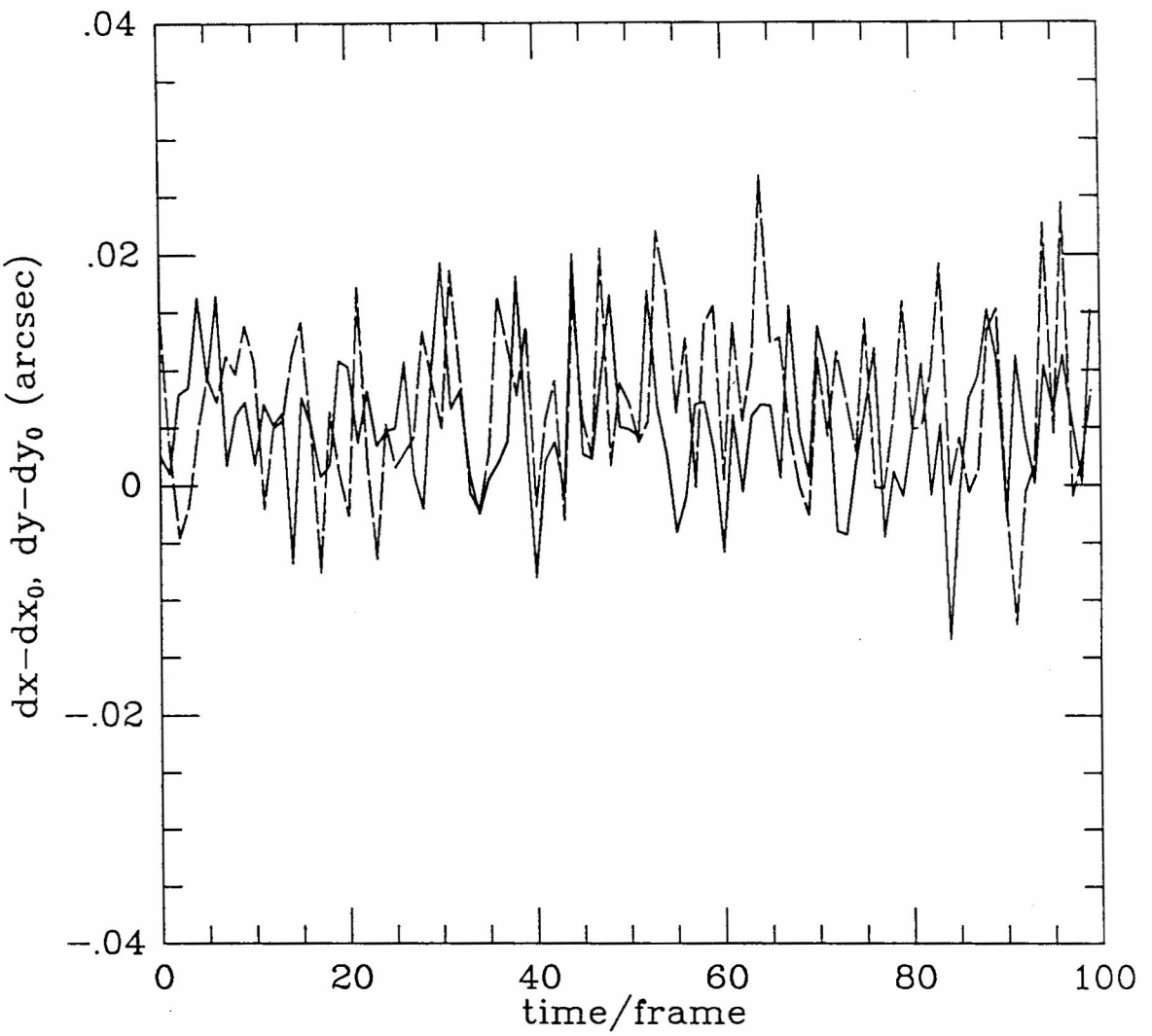


Figure III.1

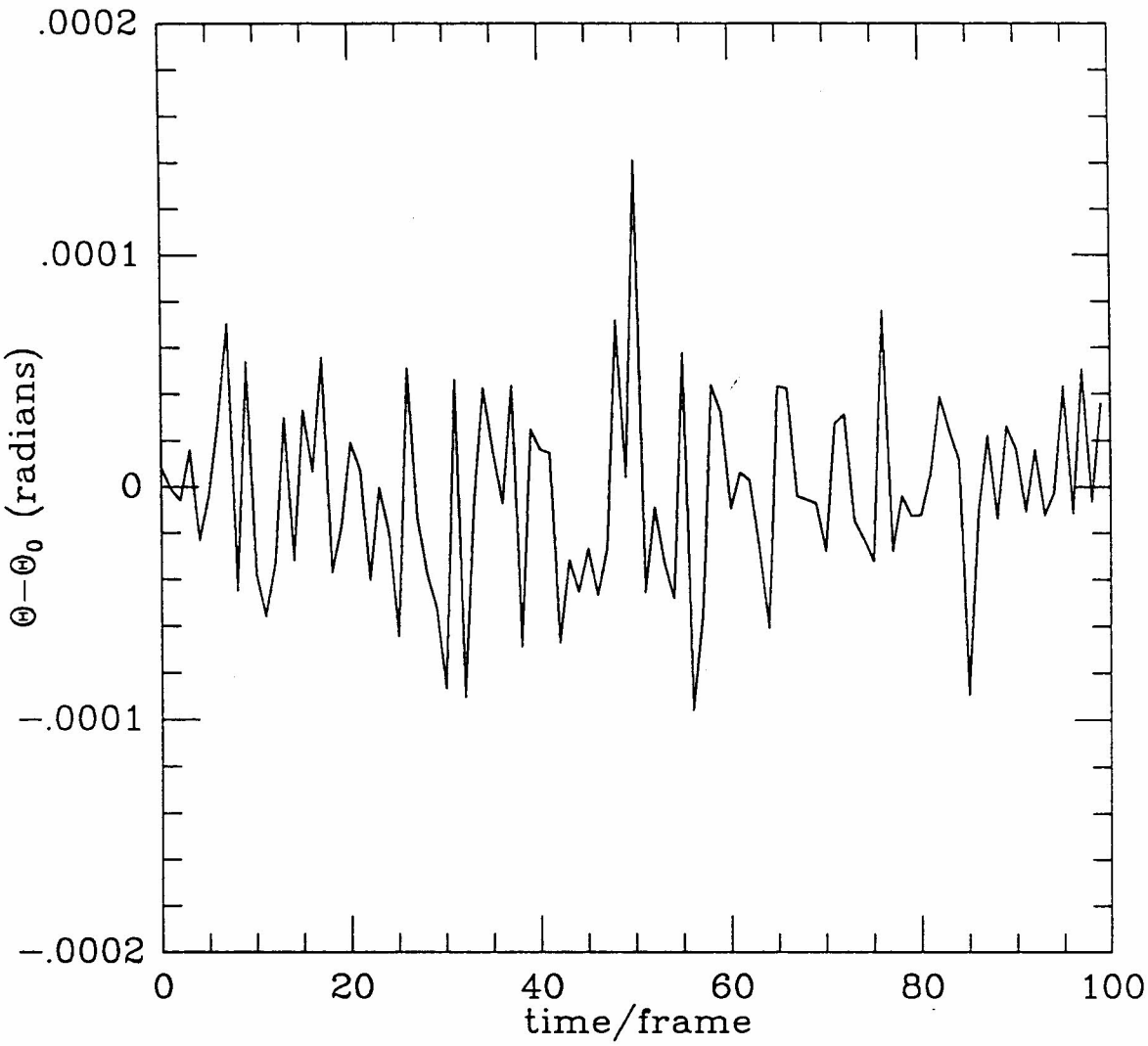


Figure III.2

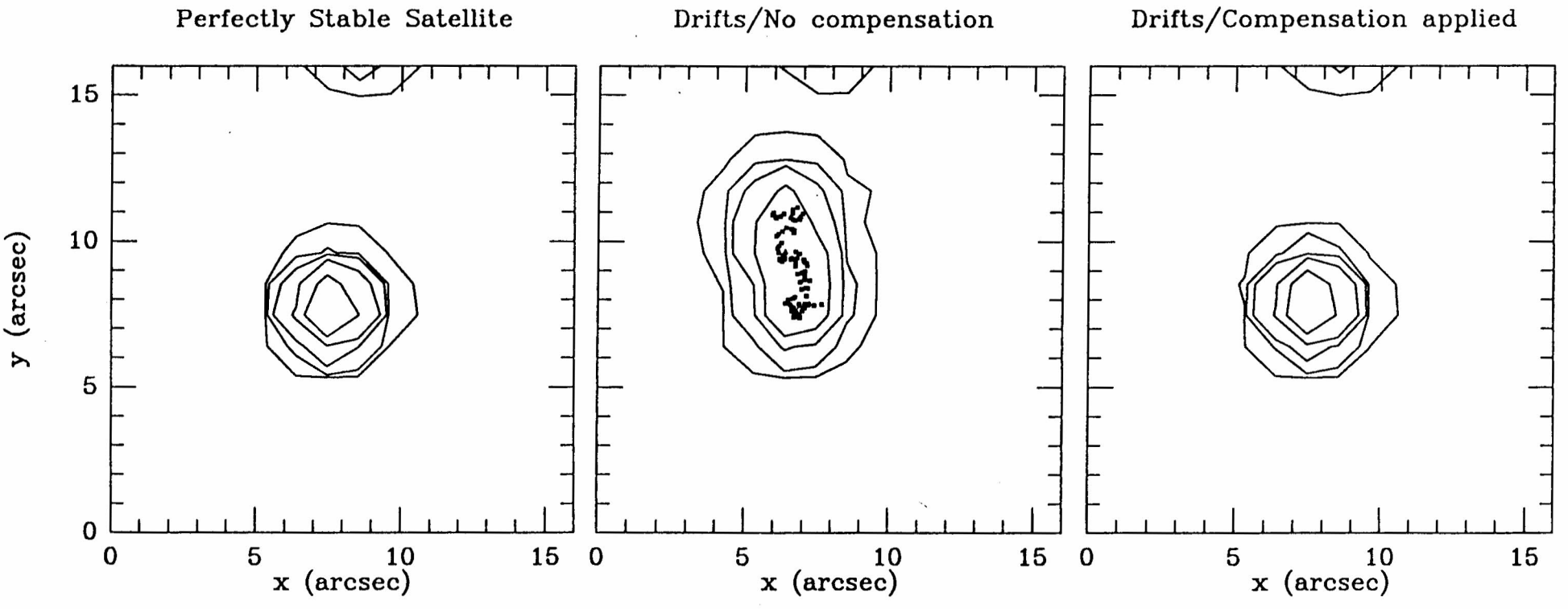

Perfectly Stable Satellite
Drifts/No compensation
Drifts/Compensation applied
y (arcsec)
x (arcsec)
x (arcsec)
x (arcsec)
0
5
10
15

Figure III.3

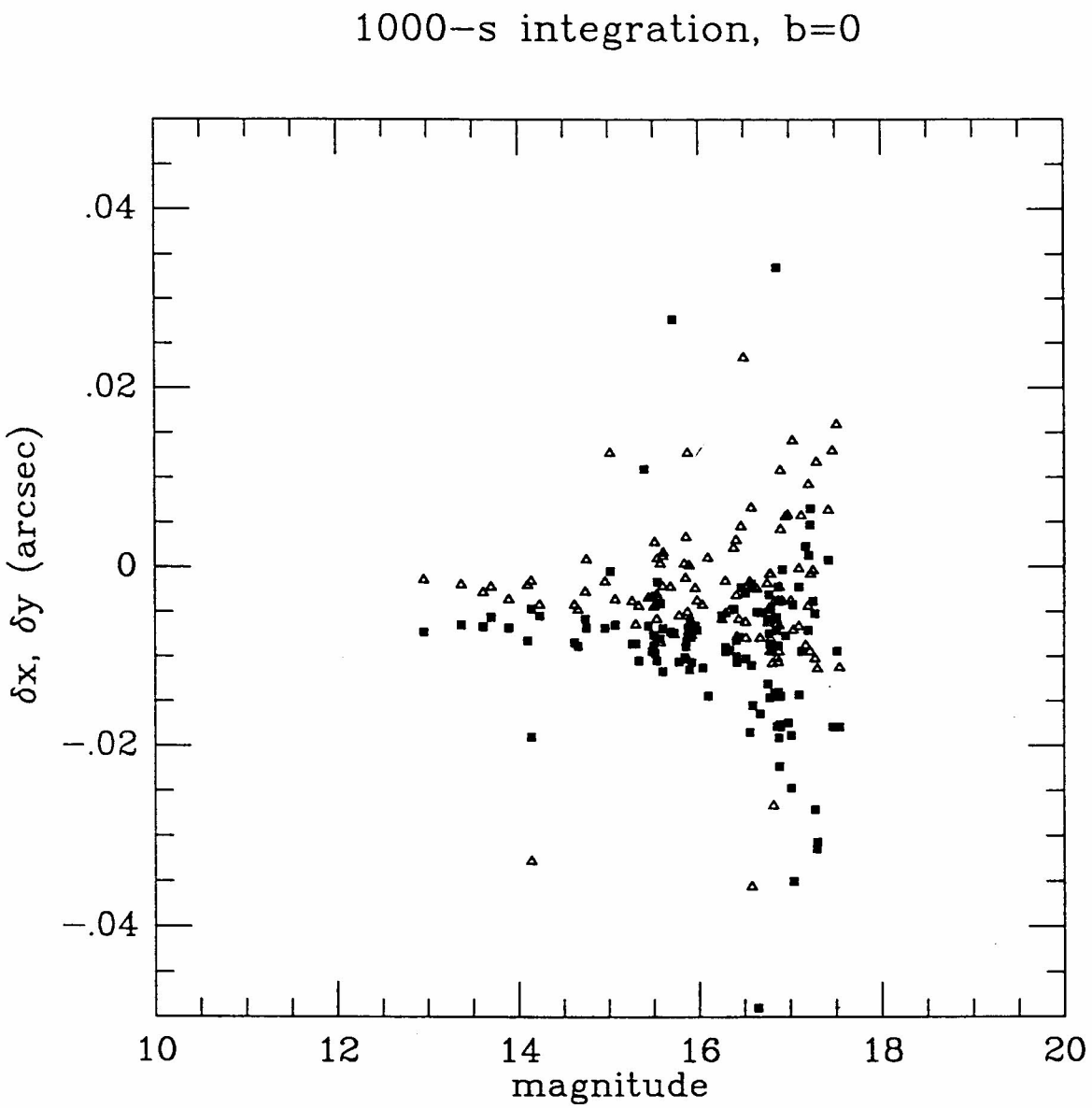

1000−s integration, b=0
.04
.02
0
−.02
−.04
δx, δy (arcsec)
10
12
14
16
18
20
magnitude

Figure III.4

IV. Shift-and-Add

IV.a. Problem

Every 10 seconds, a frame of 1024 by 1024 pixels is generated. DPU using the microprocessor will perform the necessary guide star image construction and calculate the relative aspect of the frame with respect to the reference frame. The current baseline calls for the accumulation of the information into a final image containing 512 by 512 *twixels*; four pixels add into one twixel. The problem is straightforward at first glance: we simply execute the following statement for all pixels

$$I_{mn} = I_{mn} + N_{ij}, \qquad (IV.1)$$

with $m = (i - \Delta x)/2$ and $n = (j - \Delta y)/2$ where Δx and Δy are the calculated drifts in the x and y direction. However, *there are more than 1,000,000 pixels in each frame and we have at most 10 seconds to process them.* Doing the shift-and-add to each pixels by a general-purpose microprocessor requires at least fetching numbers from two memory locations, adding two integers and writing the result to one memory location. (In addition, the threshold mode processing will also go through the same procedure of scanning through all one million pixels.)

It has be recognized that a system with the baseline microprocessor 80C86 running at 4 MHz simply does not have the horsepower to accomplish this task in 10 seconds. The problem is further complicated by the segmented memory addressing scheme of the 8086 processor.

There are the following directions we can go to attack this problem.

- Use a more powerful processor. Advantage: Versatility. Disadvantage: May not find a processor powerful enough.

- Use a dedicated processor to handle the shift-and-add task only. Advantage: Versatility. Disadvantage: Power consumption (?). Somewhat complex circuitry.

- Use a specially designed hardware coprocessor. Advantage: Room for optimization. Disadvantage: Lack of versatility. Complex circuitry design. Cost (?).

The third possibility is viable since the job of shifting-and-adding each pixel is quite simple and mundane; we are limited by quantity, not by complexity. We shall discuss various considerations for the third option in the next section.

IV.b. Shift-and-Add with Hardware Processing

Following conversations with Sandia people, it appears that a shift-and-add hardware co-processor (SAAHC hereafter) external to the microprocessor can be designed and fabricated within a reasonable bound of resources to handle the raw chore of shift-and-add. Techniques such as pipelining can be applied to reduce the effective cycles per pixels. For example, it has been estimated by the Sandia team that the shift-and-add can be accomplished in about 2 seconds: 2 μs per pixel or

8 cycles per pixel on a 4 MHz system. Even if we double the estimated processing time to 4 seconds to be overly conservative, we still have 6 seconds left to carry out the important task of analyzing the guide stars, calculating the drift and roll, performing the threshold mode processing, doing data compression and servicing the red instrument. The problem is this: *how do we formulate the shift-and-add algorithm in a fashion suitable for hardware design and implementation?*

In designing the shift-and-add algorithm, we are faced with three considerations.

• First consideration: The satellites drifts and rolls during the 100-frame integration and the FOV changes in the process. If we want to include all photons accumulated by all frames, then we need to reserve a space greater than 512 by 512 (612 by 612, say). However, the benefit in additional science is marginal. Furthermore, memory addressing will be more complicated for an array whose size cannot be expressed as a power of two. Thus it has been decided that the final image which is the sum of all 100 frames will have a size of 512 by 512. And since we will have no prior knowledge of which direction the drift is going to be in, we will use the FOV of the reference frame as the FOV of the final image.

• Second consideration: As the aspect changes, some of the frame pixels map to a region which is outside the FOV of the reference frame. We do not want to include these informations. Correspondingly, some areas in the final image which map outside the FOV of the current frame should not receive any information from the current frame. We need to take care this.

Third consideration: The memory in a computer is always stored and addressed in a linear fashion. For the two dimensional image we need to do two levels of shifting-and-adding. This is complicated by the fact that we are shifting-and-adding 2 by 2 pixels in a frame into a single twixel in the final image. In the following, we shall assume that the memory is stored in a row-major fashion, i.e., the indices of the column or the y component runs fastest. This is the manner in which arrays in C are stored.

The design and the function of the SAAHC depends on how much complexity and intelligence we are willing to build into it and how much preparation by the microprocessor we are willing to provide. There are (at least) three approaches.

1. The drifts are supplied to the SAAHC. No preparation is performed on the data in the frame. The SAAHC must have the intelligence to figure out the beginning and the end of the adding along each column and add only the pixels in the current FOV that overlaps with the reference FOV. This approach requires a lot of SAAHC intelligence. A schematic of this approach is shown in Figure IV.1.

2. The frame data are doctored by the microprocessor in that any pixels in the frame which sit outside the FOV of the final image will be set to zero. In addition, two 1024-pixels columns of padding, one on each end of the frame, will be appended to the frame. After the appropriate preparation, the beginning of the frame pixel and

the image twixel to be added as well as the length of the addition *for the whole frame* will be supplied to the SAAHC which then carries out the adding in a linear fashion in one pass. A schematic of this approach is shown in Figure IV.2.

3. The microprocessor keeps track of the beginning and the end of each column to be added. The SAAHC is only responsible for handling *one* column of frame data at a time. The microprocessor will supply the beginning frame pixel and image twixel to be added and the length of the addition *for the column* to the SAAHC. Given these numbers, the SAAHC will process the data in the column. After processing each column, the SAAHC sends an interrupt to the microprocessor and request information on the next column. A schematic of this approach is shown in Figure IV.3.

A reasonable design philosophy is this:

- We have to have the SAAHC to do the job.

- The SAAHC should be as simple as possible. Anything that can be done within the time constraint (10 s) by the microprocessor should be done by the microprocessor. Among the plausible schemes, the simplest SAAHC will be the most desirable.

Based on this philosophy, the first approach appears to demand too much intelligence in the SAAHC. We will not discuss it further. The second and the third approach appear comparable in the microprocessor requirement and the complexity of the SAAHC.

Advantages of Approach 2:

- The communication between the microprocessor and the SAAHC appears minimal. Once the necessary information is supplied to the SAAHC, the microprocessor will be freed to carry out other tasks. There will be occasional interrupts sent by the SAAHC to the microprocessor to call for threshold mode processing.

Disadvantages of Approach 2:

- The SAAHC is slightly more complicated (then that for Approach 3) since it needs the intelligence to do the double incrementing both column-wise and row-wise in order to achieve the 2 by 2 into the 1 by 1 mapping.

- Doctoring the frame memory and zeroing out the out-of-bound (the bounds of the FOV) pixels is a non-trivial job.

- Additional memory of 1024 by 2 by 4 bytes is needed. However, this additional amount of memory is negligible in the overall memory budget.

Advantages of Approach 3:

- The SAAHC is probably simpler.

• No additional memory is required.

Disadvantages of Approach 3:

• Keeping track of the beginning and the length of adding for individual column by the microprocessor is a non-trivial task. It is a more complicated algorithm than the simple doctoring in approach 2.

• It requires more communication between the microprocessor and the SAAHC. The microprocessor send information to the SAAHC about 1000 times, and the SAAHC also talks to the microprocessor about 1000 times, plus the interrupts calling for threshold mode processing.

Overall, I feel that Approach 3 has the advantage of providing the simpler SAAHC designing requirement. The doctoring in Approach 2 and the range bookkeeping in Approach 3 probably take similar amounts of processing time; one is memory-IO-intensive and the other is CPU-intensive. The communication requirement between the microprocessor and the SAAHC may or may not be an important issue. On the one hand, as long as the communication occurs on a μs timescale, this is probably not critical. On the other hand, the request for information from the microprocessor by the SAAHC will have to compete with other processes.

Current implementation in the software follows Approach 3.

IV.c. Shift-and-Add with a Microprocessor

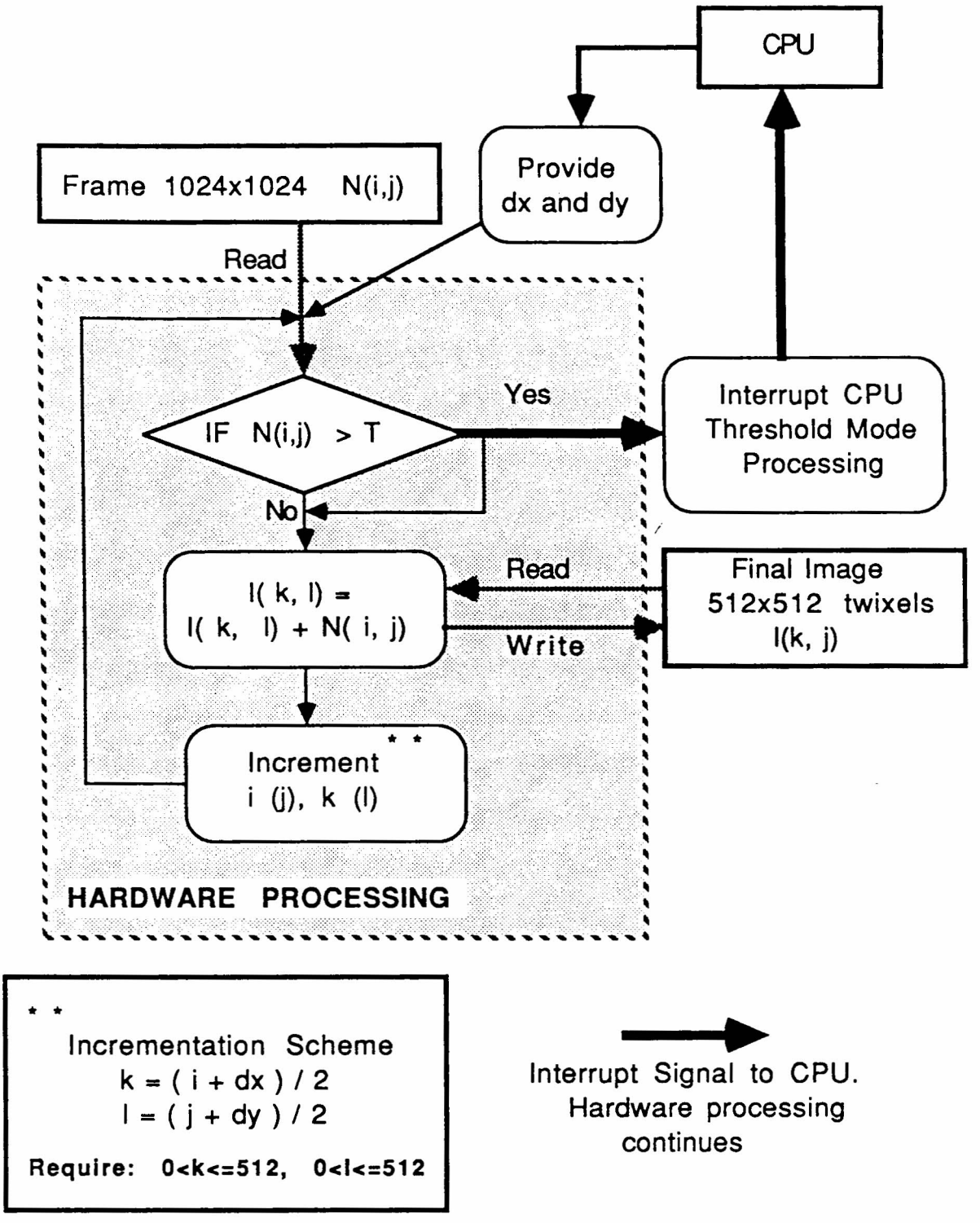
Although much of the algorithm development work discussed above has concentrated the utilization of a SAAHC, we need to keep open the possibility of doing the shift-and-add with the generic microprocessor. This factor is especially true in light of the testing of the algorithm above by HRB personnel on a Transputer running at 17 MHz. Using a scaled-down image, it was estimated that the shift-and-add of the entire 1024 by 1024 frame to the final 512 by 512 image will take about 3 seconds. Assuming the memory addressing downsizing in the testing procedure do not affect the actual performance, then the amount of processing is simply proportional to the clock rate. Based on this result, a dedicated Transputer running at $\gtrsim 6$ MHz can comfortably take care of shift-and-add task. If dedicated RAM is attached to this Transputer, then the minimum memory requirement is 3 MByte plus change. It is conceivable that this Transputer could also be used to do the important but somewhat less CPU-intensive task of data compression if we have enough CPU cycles (say 10 MHz Transputer) and memory ($\gtrsim 4$ Mbytes). The versatility of doing the shift-and-add with a microprocessor is its greatest advantage. It should be emphasize that even though a relatively powerful dedicated microprocessor is used, we should invest effort into the optimization of the algorithm at the machine code level as the payoff could be substantial.

Figure Captions for Section IV

Figure IV.1 — Schematic of the shift-and-add procedure using a smart hardware co-processor (Approach 1).

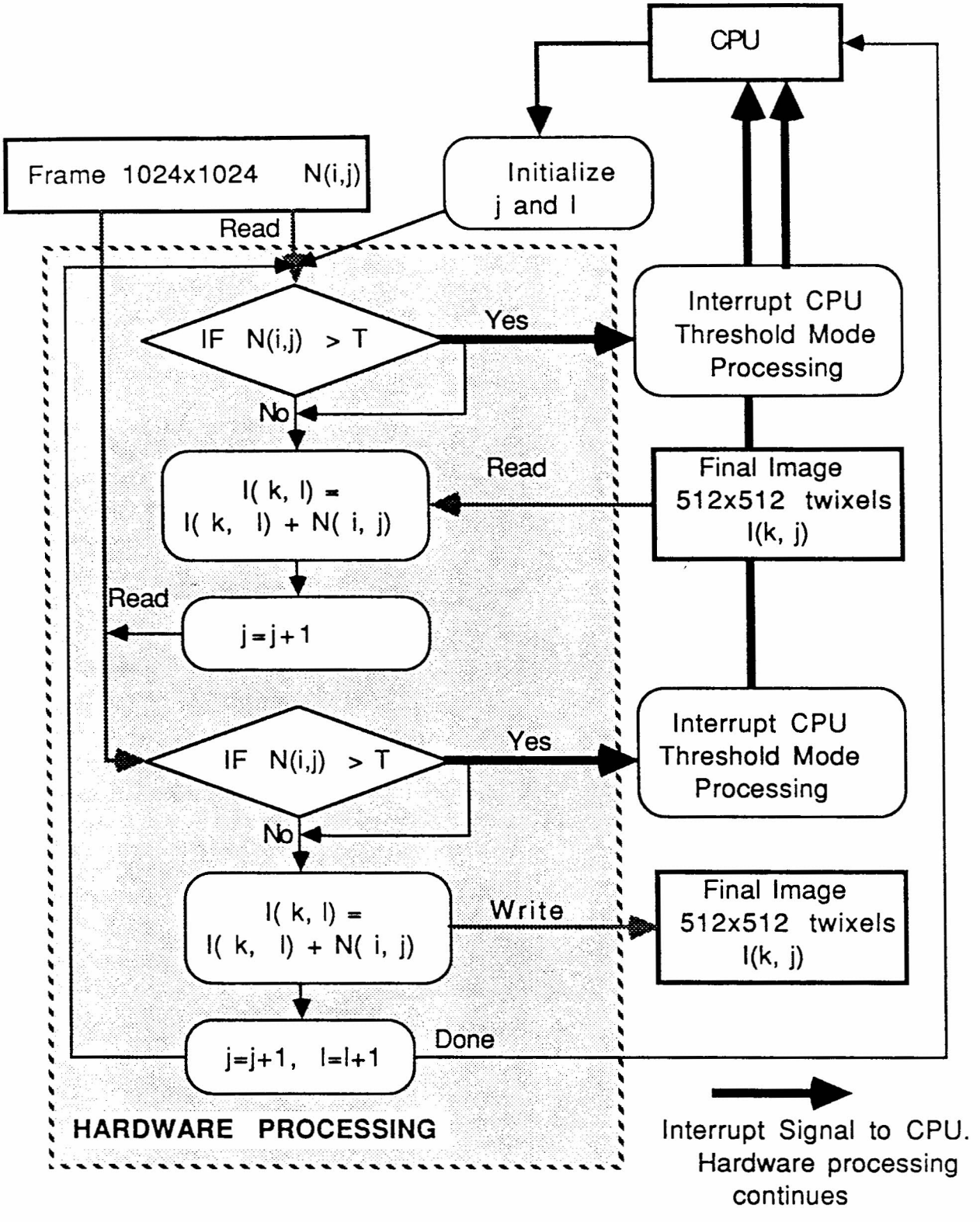
Figure IV.2 — Schematic of the shift-and-add procedure using a moderately smart hardware co-processor (Approach 2).

Figure IV.3 — Schematic of the shift-and-add procedure using a simple hardware co-processor (Approach 3).



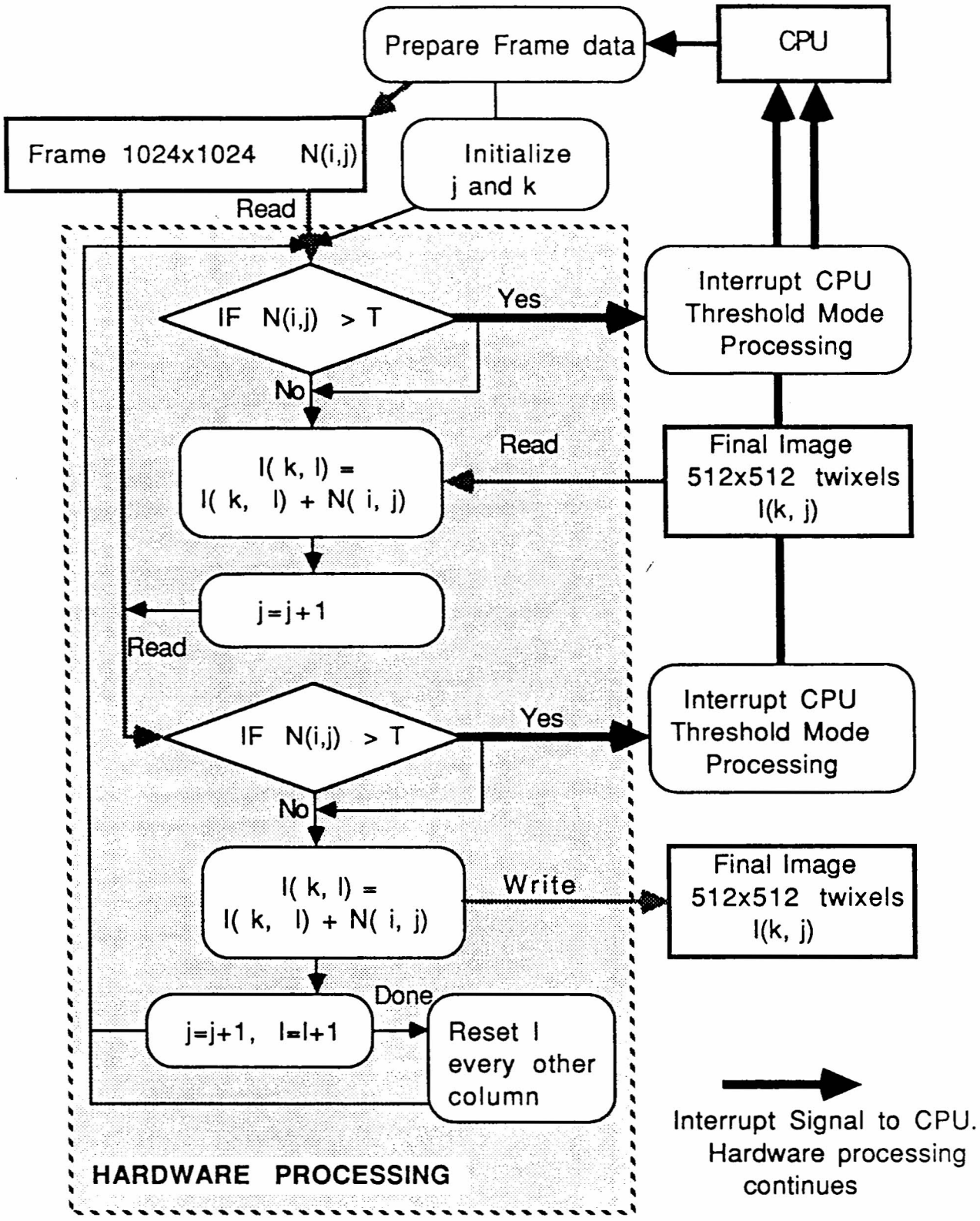
Schematics of Shift-and-Add and Threshold Mode Processing: Approach 1

Figure IV.1



Schematics of Shift-and-Add and Threshold Mode Processing: Approach 3

Figure IV.3



Schematics of Shift-and-Add and Threshold Mode Processing: Approach 2

Figure IV.2

V. Compression of Image Data

V.a. Problem

After an integration of 100 frames, we get an image which is stored in an array of 512 by 512 twixels by 4 bytes. The dynamic range of each twixel is from about 300 counts up to about 4×10^6 counts. A raw count of the information content is about to 5.5 Mbits. On the other end, we have about 1000 sec to transmit this image. With a telemetry rate of 2 kbits/s and a 85% duty cycle, we can only transmit about 1.7 Mbits of data. Thus we need to compress the data by a factor of greater than 3. To devise a compression scheme best suited for our purpose, we need to take into account several factors jointly.

- We want to maximize the compression ratio.
- We want to retain the maximum amount of information. A fully reversible scheme should be used if possible.
- We want to maximize the tolerance of the compression-transmission-decompress scheme to telemetry errors.
- The job must be done within the constraint of memory allocation and processing power.

At this point, the processing power constraint is probably not critical since we will have about 1000 sec to carry out the compression task even though the effective duty cycle might be low.

In the following, we will describe a promising candidate for a reversible compression scheme which can deliver a compression ratio close to (but not quite achieving) the requirement. This is the Variable-Block-Tiered-Word-Length (VBTWL) Scheme.

V.b. Data Compression

In general, a compression scheme contains two major parts: decorrelation and coding. The decorrelation procedure removes redundant information content. We want to maximize the information content taken out by the decorrelation. What's left is the information content which is random. Then the coding scheme seeks to minimize the number of bits needed to represent this random information.

Prior to the decorrelation procedure, the entire dat set will be segmented into blocks which contains packets of data; each packet corresponds to a twixel in the final imgae from the image mode observation. The number and size of the blocks can be adjusted so as to give us the optimal performance. The best segmentation scheme depends on the characteristics of the data and other considerations. In principle, the segmentation of data into blocks will allow us to search for the best decorrelation and optimal compression scheme/parameters individually, i.e., we can look for local optimum. On the other hand, segmentation also incurs overhead; the finer the

segmentation, the greater the overhead. Overall performance of the compression scheme requires a good balance between minimizing the overhead and obtaining the best optimization of local compression within each block.

For the individual blocks, we do the decorrelation procedure by making a guess of what the value of each data packet is; this is called the predictor. The difference between the actual data and the predictor is the reduced data. If a good predictor is used which removes the correlated portion of the data, then the reduced data will be mostly random. One way to think of this is the analogy of the zeroth-order solution (predictor) and the first-order perturbation (reduced data). In general, the zeroth-order solution is simple and the first-order perturbation is small. Ths simplicity of the predictor and the smallness of the magnitude of the reduced data allow us to compress the information content into a number of bits smaller than in the original format.

After the decorrelation, we code the reduced data into a stream of bits. There are many possibilities of coding as described in the report of Adaptive Data Compression System Study prepared by the Satellite International Limited (SIL) under an ESA Contract. This report was dated July 1988 and is available within the XMM/OM consortium. Depending on the characteristics of the random data, different coding schemes are optimized for different applications. The coding mechanism of the VBTWL scheme discussed here, developed independently at Penn State, is a generalization of the Variable-Block-Word-Length (VBWL) scheme described in the SIL report.

V.c. Characteristics of Image Data

The final image from the blue detector is stored in a two-dimensional 512 by 512 array. There will be point sources (stars) and extended sources in the image. But most of the photons are probably coming from the background which, in white light, contributed to about 0.3photons/s/arcsec^2, or 300 photons/twixel/1000s. The background statistics is Poisson which, in the large number limit, is approximately Gaussian. Since the image is two-dimensional, the correlation within a block of data is expected to be greater if the block corresponds to a square in the image (e.g. a 16 by 16 twixel area) than the case of a line (e.g. 256 by 1 twixels column).

For a block mostly dominated by the background, it is easy to argue that the mean count rate is a very good predictor; any deviation from that is random. Another possibility is the minimum count rate within the block. For a block which contains some kind of structure above the background, neighboring twixels are likely to be correlated. Using the count of a neighbor as a predictor may provide good decorrelation in these cases. Thus, we have three possible decorrelation mechanisms. We decorrelate the data within the block by coding the information in terms of X_{ij} where

$$X_{ij} = I_{ij} - P_{ij}, \qquad (V.1)$$

and the predictor can be one of the following three:

- $P_{ij} = \bar{I}$, the mean within the block,
- $P_{ij} = I_{\min}$, the minimum within the block,

or

- $P_{ij} = I_{i(j-1)}$, the count in the previous neighbor.

The reduced data X are then fed into the coding scheme.

V.d. Tiered-Word-Length Coding

In the binary representation, the content of useful information of an integer is the number of bits needed to represented the number. For example, the number 7 needs 3 bits. If we use 16 bits to represent the number 7, then we are in a way wasting 13 bits. However, we only know we need 3 bits *after* we know we want to express the number 7. The information that we have a 3-bit number must also be conveyed somehow. An analogy is the exponent and mantissa of a floating number which express the magnitude and the value of the relevant number. In the VBWL scheme described in the SIL report, the coding is done by examining the maximum number of bits required to express any packet of reduced data within the block. After that, the reduced data is expressed uniformly in the same number of bits, irrespective to its actual value. The Tiered-Word-Length coding differ from the VBWL scheme in that the coding is adaptive. Current implementation of VBTWL allows for three tiers; coding with a smaller number of tiers is a subset of the three-tier structure. For example, the original VBWL can be encompassed in the 3-tier VBTWL, with the constraint that all three tiers must be the same.

The three-tier coding mechanism is governed by the following rules.

- Tier 1: Any number that can be expressed in n_1 bits, other than the two flags f_2 and f_3 discussed below, will be coded in n_1 bits.

- Tier 2: Any number that cannot be expressed in n_1 bits but can be expressed in n_2 bits will be coded in n_2 bits *preceded by a unique flag f_2 which is n_1 bits long.* Let's suppose there are N_2 of them.

- Tier 3: Any number that cannot be expressed in n_2 bits but can be expressed in n_3 bits will be coded in n_3 bits *preceded by a unique flag f_3 which is n_1 bits long.* (The number n_3 is in fact the maximum number of bits required to express any piece of reduced data within the block.) Let's suppose there are N_3 of them.

- Flags: The two numbers adopted as the distinct n_1-bit long flags f_2 and f_3 are considered to be members of Tier 2 and coded accordingly.

In a block of N_B packets of data, the number of bits in the data stream coded following these rules are

$$M = N_{header} + n_1 N_B + n_2 N_2 + n_3 N_3, \qquad (V.2)$$

where N_{header} is the number of bits in the header which contains the decorrelation and coding information such as the predictor scheme, the predictor and the tier structure. Redundancy check can also be incorporated into the header to help ensure data integrity. Aided by the information provided by the fixed-length header, the compressed data stream can be uniquely decompressed to retrieve the full information content of the block. This is a fully reversible scheme. In general, there can be any number of tiers. The optimum is determined by the balance between the increased efficiency in expressing numbers, growing size of the header, and the greater complexity, as the number of tiers grows bigger. At this moment, the two-tier or three-tier structures appear to provide a good balance between compression ratio, header size and complexity.

Figure V.1 shows the conversion of a real data stream of final image to a compressed data stream. The processing flowchart is shown in Figure V.2. An alternative to the flagging of higher-tier data packets is to construct a "road map" of the higher-tier number outside the main data stream and eliminate the use of the flags. This road map will be a part of the header. Whether such a scheme yields a better performance in compression ratio remains to be investigated.

V.e. VBTWL – Current Implementation

Block-Wise Compression

Given a block of reduced data processed by the decorrelation predictor P, we can apply a TWL coding mechanism parameterized by (n_1, n_2, n_3) and obtain a length of the compressed data M. It is clear that n_3 is determined by the reduced data: it is the number used in the conventional VBWL scheme. On the other hand, n_1 and n_2 are completely arbitrary except for the constraint that $n_3 \geq n_2 \geq n_1$. With n_1 and n_2 as free parameters, we can search for the combination (n_1, n_2) which yields the minimum M. After applying this procedure of optimizing (n_1, n_2) to the three decorrelation mechanisms with their respective predictors, we arrive at a set of parameters $(P; n_1, n_2, n_3)$ which yields the minimum number of bits for the current block. Preceded by the appropriate header, the data stream is then supplied to the downlink mechanism.

Block Shape and Sequencing

As discussed earlier, the decorrelation is probably most efficient where the block structure is two-dimensional; i.e. a square is better than a rectangular which is better than a line. Furthermore, a square block has the advantage of being more tolerant to telemetry error. Suppose a square 16 by 16 block of data is corrupted beyond recovery and each bad twixel will affect two twixels on each side, then the final data will be corrupted in square of 20 by 20, or 400 twixels will either go bad or be affected. In contrast, if the corrupted 256-twixel block is taken from the final image in a 1 by 256 column, then there are altogether 1280 twixels which either go bad or are affected. From the point of view of data integrity and resilience, a square block is definitely more favorable.

Suppose we are taking blocks of 16 by 16 twixels, then there will be 32 by 32 blocks in the 512 by 512 final image. We can also devise different sequences of extracting and compressing the blocks which provides operational advantages. The simplest way of sequencing the block is to do a column-wise (or row-wise) scan from, say, the lower-left corner to the upper-right corner. With a simple algorithm, however, we can trace out the blocks in a spiral fashion as shown in Figure V.3. In the spiral sequence, the central portion of the image, presumably of the highest quality, will have the highest priority in being processed (compression and transmission). Since the blocks are transmitted in a sequence of nearest neighbors, the real time display of images follows a continuous development. Most importantly, if for whatever reason the telemetry stream is disrupted, the portion of the image with the highest quality will have the highest priority for processing. Salvaging a bad situation is easier this way. In principle, we can also apply the spiral sequencing to the processing of twixels in individual 16 by 16 blocks. However, the advantage is less apparent and it will require more processing power. At this point, the processing of twixels within a block is done in a row-major scanning fashion.

V.f. Performance and Operational Issues

Table V.1 lists the size of the compressed data stream of simulated 512 by 512 final images in white light with various stellar density. We have listed three columns of results with VBTWL and one with VBWL. We have also listed four columns giving different block sizes. Furthermore, the size of the compressed data stream also depends on the size of the header. It appears that 40-bit header is the minimum for the 3-tire VBTWL and a 64-bit header will provide great versatility and redundancy check. For VBWL, a 32-bit header for each block of data appears sufficient.

TABLE V.1

Compression of Data Stream

Coding Scheme Star Field Header Size (bits)	VBTWL G. Equator 40	VBTWL G. Equator 64	VBTWL Average 64	VBWL G. Equator 32
Block Size	(Data stream size in bits)			
4	2339743	2732959	2713467	2209984
8	1944781	2032085	1991070	2405056
16	1861496	1886072	1856473	3175168
32	1880013	1886157	1885802	3838976

If a 2 kbps telemetry bandwidth is allocated and the image data is transmitted with a 85% duty cycle, then we can transmit 1.7 Mbits. The numbers given in Table V.1 shows that we are close but not yet there. I think we can take the following approaches listed in the order of their respective desirability.

(1) Ascertain a greater telemetry bandwidth. Even a meager 10% to 20% increase will put us in a very comfortable position.

(2) Extend the integration time, thus increase the number of bits that can be transmitted. On the other hand, the size of the compressed data stream will probably also increase, at a smaller rate though.

(3) Discard part of the image to conform with the telemetry/time constraint. The spiral pattern processing is intended to minimize the effect of such truncation by making sure that any truncation will result in the loss of lower quality portion of the image.

(4) Adopt a non-reversible scheme.

(5) Continue to search for a more efficient reversible compression scheme. See discussions below on the prospect of substantial improvement in compression over current implementation. (As a side note, it has be determined that a double-differencing scheme does not provide improvement in the compression ratio. The decorrelation scheme 3 described in *V.c.* is single-differencing. The double-differencing scheme takes the difference between the differences of consecutive twixels as fundamental quantities for compression.)

Even though we have discussed three different decorrelations mechanisms, there could be other viable schemes and we should to keep our minds open on these possibilities. Furthermore, the code should be implemented to allow the disabling any particular decorrelation mechanism if it is deemed uneffective. For example, it has been discovered that in compressing a typical star field, less than 3% of the blocks use the single-differencing scheme as the optimal method. In processing, each decorrelation mechanism requires the similar amount of CPU cycles. Thus, if we are starved for CPU cycles, disabling a decorrelation could yield tremendous benefit at a price of slightly lower compression efficiency.

V.g. General Issues of Data Compression

It is helpful to establish some kind of benchmark where the performance of the compression algorithm can be compared to. In the limited research that went into this project, I have not encountered any such estimate in the literature. However, there appear to be some first principle arguments that one can apply and derive a simple-minded estimate of the "information content" of a data set.

As described in *V.b.*, the goal of a good compression scheme is to (1) maximally remove the redundant component of the data set and (2) minimally formulate the random component of the information. In our application, the field of view is dominated by twixels which contain only photons from the background.

At a level of average 300 photons/twixel, the distribution is well approximated by a Gaussian with a standard deviation of about $\sigma_N \sim \sqrt{300} \sim 18$. If the image contains *no* stars, then more than ninety-nine per cent (99%) of the twixels will be within $\pm 3\sigma$ of the mean (300 counts/twixel). To express the random information in each of these 99% of the twixels, we need about

$$\log_2(2 \times 3 \times \sigma_N) \sim 7 \text{ bits} \qquad (V.3)$$

per twixel. It is difficult to see how one can devise a coding scheme which can, in an average sense, express the non-reducible random portion of the data in a number of bits much smaller than this. Thus, the total number of bits required for the 512 by 512 image is about

$$N \sim 512 \times 512 \times \log_2(2 \times 3 \times \sigma_N) \sim 1.8 \text{ Mbits.} \qquad (V.4)$$

We now formulate these considerations into the following conjectures:

Conjecture 1: An optimal reversible compression algorithm will yield a data stream containing approximately N bits, where

$$N \sim N_p \times \log_2(\psi). \qquad (V.5)$$

Here N_p is the number of packets of data, ψ is a measure of the randomness of the data.

Conjecture 2: In the case of a data set dominated by a random component which follows Gaussian statistics, then $\psi \sim 6\sigma_N$, or

$$N \sim N_p \times \log_2(6\sigma_N), \qquad (V.6)$$

where σ_N is the standard deviation of the background.

We now discuss the application of these conjectures to our results.

First, the estimate given in equation (*V*.4) is very close to the number of the we obtained in *V.f.* This indicates that the VBTWL scheme is probably very close to being the optimal for our application, if the above conjectures are correct. This also means that attempts to further increase the compression ratio will probably not be cost-effective.

Second, the estimate (*V*.6) does not have any explicit dependence on the stellar density. In reality, the stellar density will no doubt contribute to the size of the compressed data stream. However, it can be viewed as a higher order perturbation to equation (*V*.6). For example, in Table V.1 the VBTWL scheme yields compressed data streams of size 1.89, 1.86 from simulated images in white light with a stellar density corresponding to the galactic plane and an average FOV; a galactic pole star field can be compressed into about 1.81 Mbits. In contrast, the background

contributes about 28%, 48% and 86% to the total number of photons, respectively. The weak dependence of the size of the compressed data stream to the stellar density is apparent.

Finally, equation (*V*.6) can be used as a rule of thumb for future design. For example, the question has been raised whether we want to process and transmit both blue and red images at the same time, if both can be fitted into the allotted telemetry bandwidth. (Among other things, this will have an impact on the memory requirement and CPU horsepower.) For a compressed data stream with a size half that allowed by the allotted telemetry bandwidth, i.e., $N \sim 1$ Mbits/1000 s, equation (*V*.6) requires that the background be of order 6 counts/twixel. In other words, we need a narrow band filter which transmits about 2% of the photons. In view of the recent discussions (Lumb) on the desirability of the greater usage of narrow band filter in order to extend the life time of the micro channel plates, the double image processing/transmission mode probably deserve more consideration.

Figure Caption for Section V.

Figure V.1 — Schematics of the Variable-Block-Tired-Word-Length data compression scheme. Each rectangle brick represents a packet of data. The accumulated number of bits is shown at the lower right corner of each packet. The real data (in the first row) is first passed through the decorrelation procedure which yield the second row of reduced data X. As an example, the packet is labeled $t1$ if the reduced data is in tier 1 and so on. The scheme will search for the optimal combination of n_1, n_2 and P which yields the minimum number of bits in the final data stream. The Tiered-Word-Length coding scheme converts row 2 into a stream like row 3, with unique flags of n_1 bits leading a higher-tier representation of data. The packets containing the flags are highlighted. As in rows 1 and 2, the accumulated number of bits is given. In this example, the data are compressed from 256 bits to 116 bits (plus header).

Figure V.2 – Flowchart of the Variable-Block-Tiered-Word-Length Scheme. The optimization procedure is not shown.

Figure V.3. — Sequencing of block of image data. We process the block of image at the center of the final image first and proceed in a spiral fashion. See text for a discussion on the advantage of this method of sequencing.

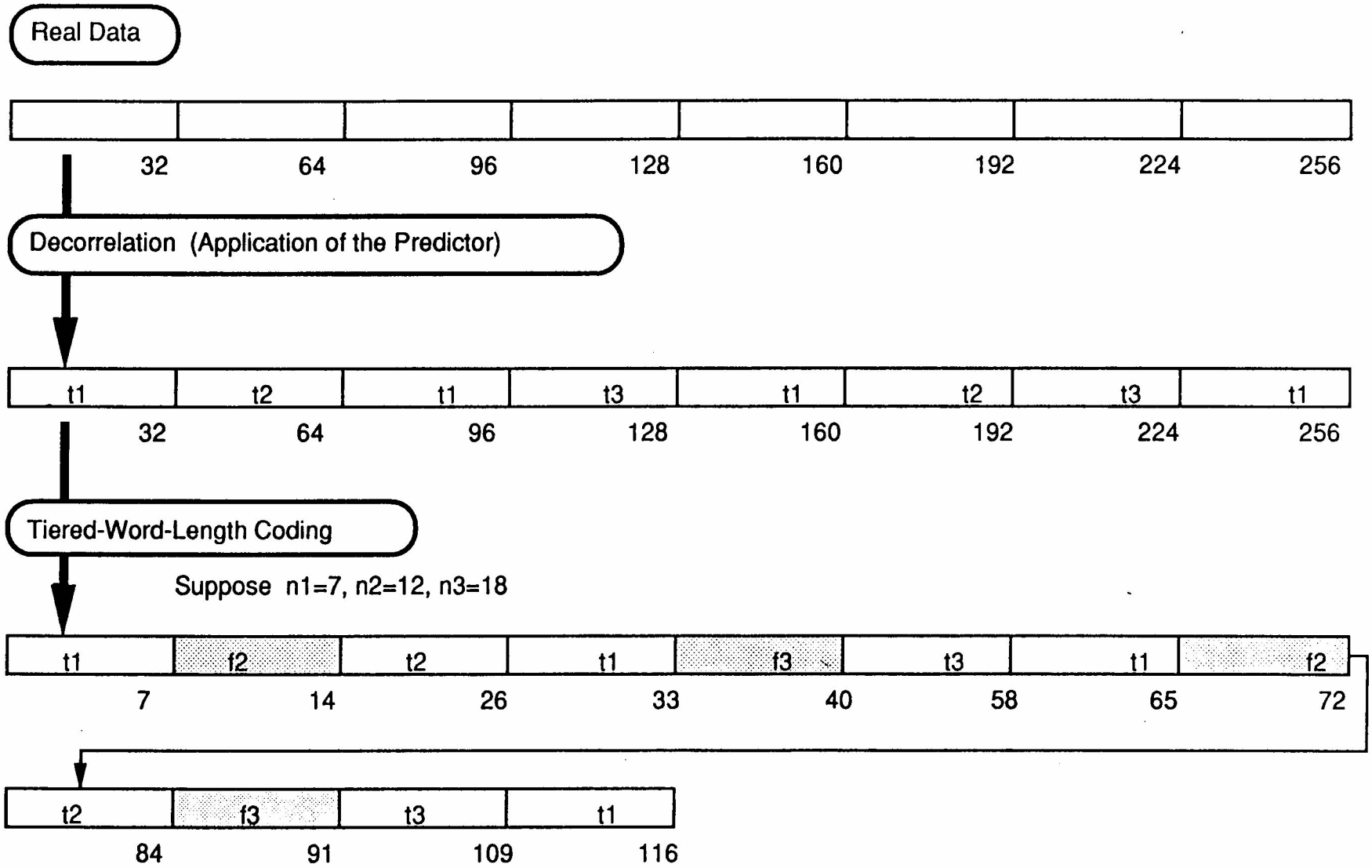
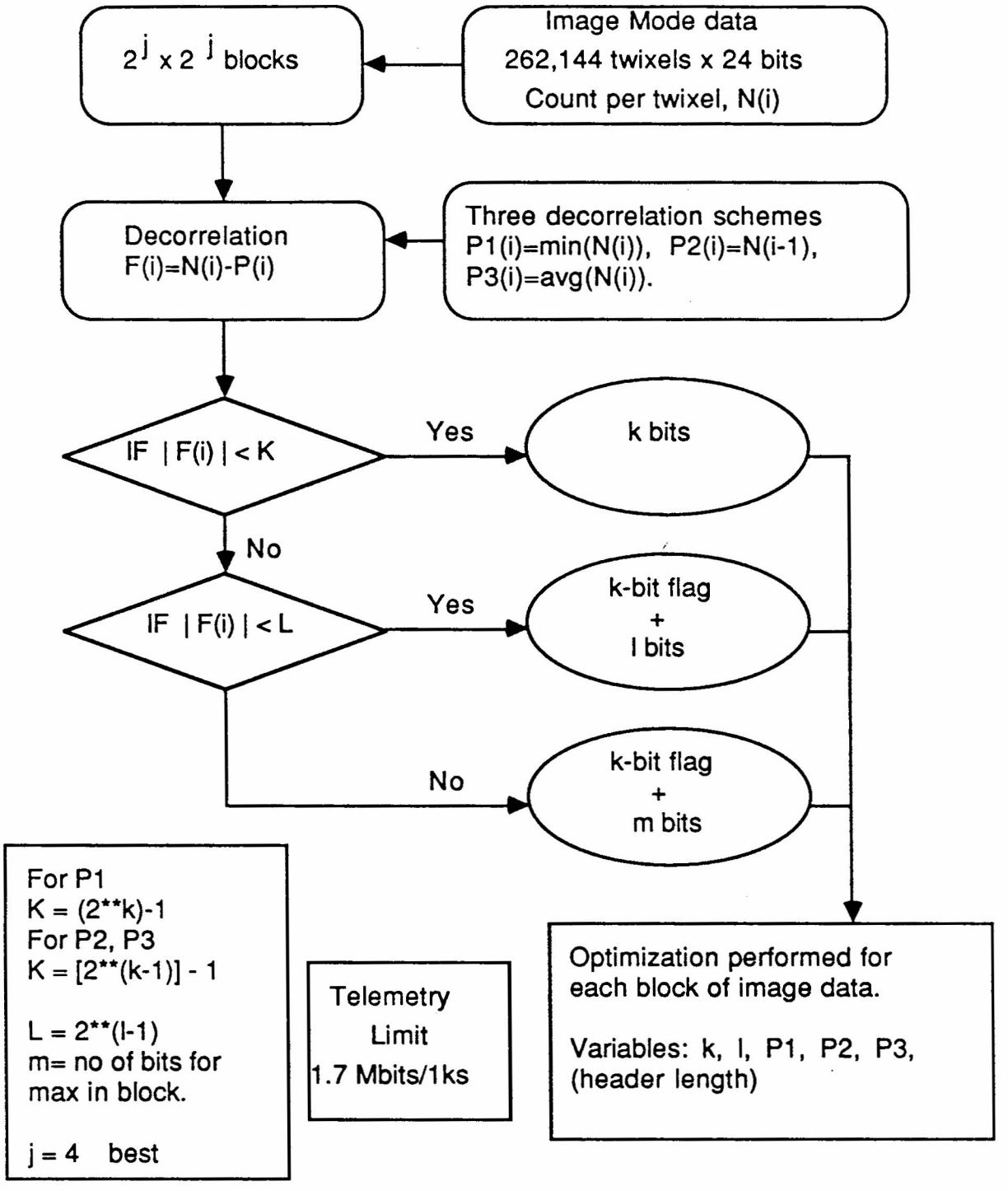


Figure V.1. Schematic of Variable-Block-Tiered-Word-Length Compression Scheme



Schematics of Variable Block Tiered-Word-Length Compression

Figure V.2

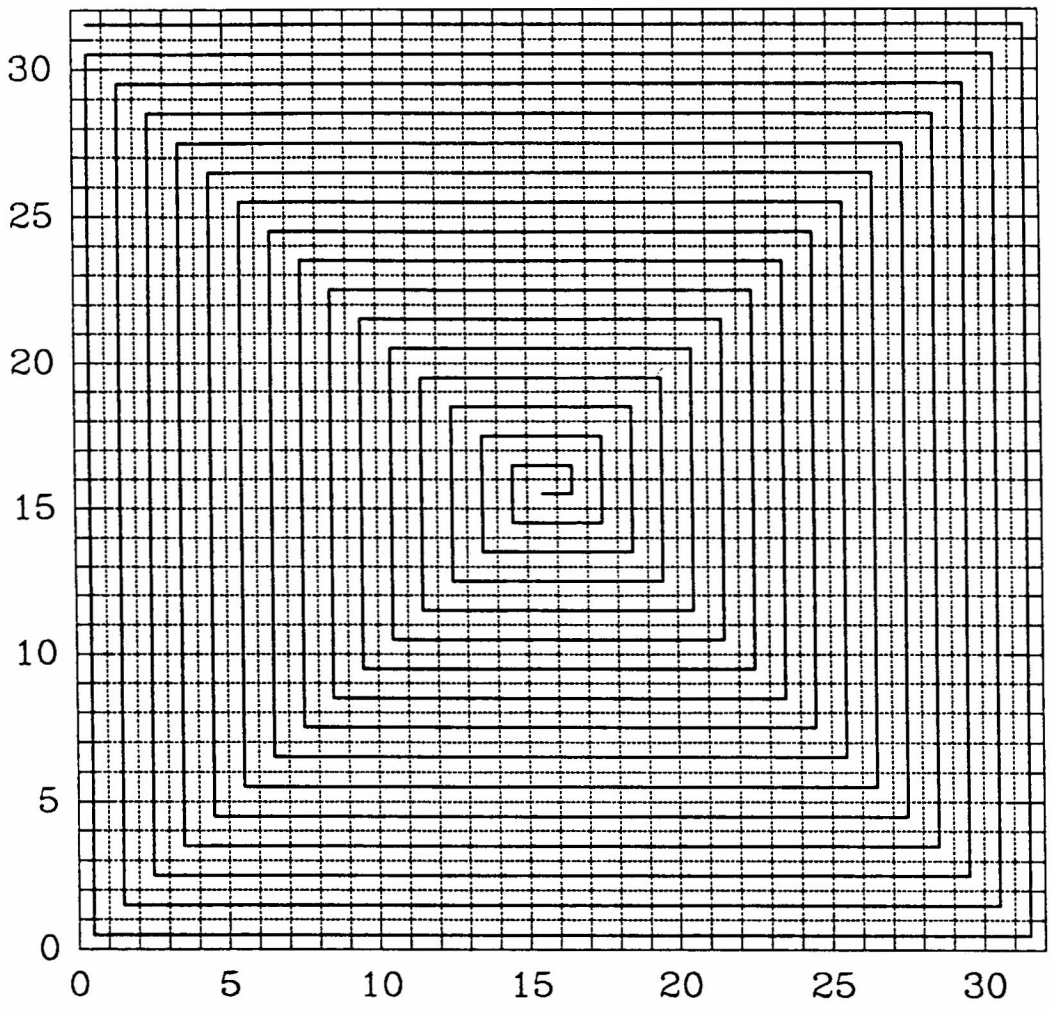


Figure V.3

VI. Accounting

In this section, we discuss the accounting issues of the algorithm discussed in the previous sections. We shall concentrate on two aspects: memory and CPU cycles. We will not worry about issues such as the real estate on the DPU board, power consumption and cost.

VI.a. Memory Requirement

Listed here are only the "big-ticket" items in the overall memory scheme for the entire DPU. Memory requirement which is less than, say 100 Kbytes, will not be discussed here; these usages are covered under the operation memory which is allocated a generous 2 Mbytes. There might be a number of them and need to be look at in the final analysis. Since the coupling of the observing modes of the blue and red instruments has not be solidified yet, I have included a separate subsection on the memory requirement of the red instrument.

Basic operation memory requirement

(1) The current working frame. This is the frame being processed for guide star location, tracking and shift-and-add. This memory requirement is fixed and cannot be reduced at all:

$$S_{iw} = 1024 \times 1024 \times 2 \text{ bytes} = 2\text{Mbytes}. \qquad (VI.1)$$

(2) The current collecting frame. This is the frame which is being written into by the blue detector contemporaneously to the processing of the current working frame above. This memory requirement is fixed and cannot be reduced at all:

$$S_{ic} = 1024 \times 1024 \times 2 \text{ bytes} = 2\text{Mbytes}. \qquad (VI.2)$$

This memory is also being read from by the fast mode processing.

(3) Memory of the compressed data to be downlinked. In principle the compression can be done on a on-demand basis, i.e., when the ICU asks for more telemetry stream, the DPU compresses the data and deliver it on the spot. However, I will feel much more comfortable that we maintain a FIFO stack of compressed data on the DPU. When the telemetry request from the ICU is received, then the operation is a simple read-and-write on the part of the DPU. On the other hand, as long as the stack is not overflowing, the CPU can compress the data in any free cycle it can find and write to the stack. A comfortable size of the FIFO stack is 1000 sec worth of data, or

$$S_{tm} = 2000 \times 1000 = 2 \text{ Mbits} = 250 \text{ kbytes}. \qquad (VI.3)$$

(4) Operation Memory. This includes the on-board software itself (and maybe a backup copy), the bright star and the guide star catalog, the drift record,

a generous stack for the processing stellar image construction, tracking, shift-and-add and compression as well as all other temporary storage space. Without a detailed accounting of the memory requirement, let's allocate a generous 2 Mbytes for this use:

$$S_o = 2 \text{ Mbytes.} \qquad (VI.4)$$

(Blue instrument) mode-specific memory requirement

(1) Threshold mode – The data generated by this mode is very little.

(2) Image mode – We want to store the final image in a memory of

$$S_{im} = 512 \times 512 \times 4 \text{ bytes} = 1 \text{ Mbytes.} \qquad (VI.5)$$

From the operational standpoint, we can get by with an allocating this much memory for the final image if we can compress the image into S_{tm} in about 10 second during which the reference frame of the subsequent 100-frame integration. However, it is highly unlikely that the compression can be accomplished within 10 s. Thus, we need to devise a memory scheme similar to the tracking procedure where two copies of the images are kept at the same time. In the compression procedure, we will have a copy S_{im} which is being written into by the shift-and-add. At the same time, we will have another copy S'_{im} which was produced by the previous integration. Overall, we need

$$S_{im} + S'_{im} = 2 \text{ Mbytes} \qquad (VI.6)$$

for storage and processing of the final images.

(3) Fast Mode – The memory requirement for the fast mode is somewhat uncertain. In the baseline design, the fast mode will be operating mutually exclusively from the image mode. Thus we can use the memory space allocated for the final images in the image mode S_{im} and S'_{im} for use by the fast mode. The allocation of 2 Mbytes appears sufficient for the use of fast mode.

Memory requirement for the red instrument

(1) Image frame from the red instrument. The red instrument will produce a frame for about every 200 seconds. For a conservative estimate, let's say the red CCD produces frame of 512 by 512 by 4 bytes, i.e. we need

$$S_{rf} = 512 \times 512 \times 4 \text{ bytes} = 1 \text{ Mbytes.} \qquad (VI.7)$$

(2) One additional red frame for cosmic ray removal. To carry out the cosmic ray removal task, we need at least one other copy of the red frame for comparison, i.e. we need

$$S'_{rf} = 512 \times 512 \times 4 \text{ bytes} = 1 \text{ Mbytes.} \qquad (VI.8)$$

Furthermore, we want to keep an array which keeps tracks of red pixels that was contaminated by the cosmic ray, the so-called C array:

$$S_{cc} = 512 \times 512 \times 2 \text{ bytes} = 500 \text{ kbytes.} \qquad (VI.9)$$

(3) Final image of the red instrument. The red frames will be accumulated in a red final image which requires

$$S_{ri} = 512 \times 512 \times 4 \text{ bytes} = 1 \text{ Mbytes.} \qquad (VI.10)$$

(4) Threshold mode of the red instrument – The threshold mode for the red instrument should not require much memory.

(5) Fast Mode for the red instrument – If the fast mode of the red instrument is mutually exclusive from the image mode, then the memory allocation above for the red image mode should be sufficient.

Final tally

The major determining factor of the total memory requirement depends on whether or not we do simultaneous image mode observations with the blue and red instrument.

If we do not do the image mode observations simultaneously with the two instruments, then the amount of memory needed is approximately

$$S_{total,1} \simeq S_o + S_{iw} + S_{ic} + S_{rf} + S'_{rf} + S_{cc} + S_{tm} \simeq 9 \text{ Mbytes.} \qquad (VI.11)$$

It is conceivable that a judicious design in memory could cut S_o down to 1 Mbytes. This will conform to the baseline design of 8 Mbytes on-board random access memory. Note that we have only added S_{rf}, S'_{rf} and S_{cc}. The same memory space can be used by S_{im} and S'_{im}, as long as we don't do simultaneous imaging. However, we then have to be careful in accounting for the memory needed for the fast modes of the two instrument.

A design which leaves greater margin of safety is to assume that we want to leave open the possibility of doing simultaneous imaging. In this case the total memory requirement, taking into account the big-ticket items are

$$S_{total,2} \simeq S_o + S_{iw} + S_{ic} + S_{im} + S'_{im} + S_{rf} + S'_{rf} + S_{cc} + S_{ri} + S_{tm} \simeq 12 \text{ Mbytes.} \quad (VI.12)$$

It appears that a total of 12 Mbytes of on-board random access memory could probably be sufficient. A total of 16 Mbytes will be leave more margin though. Note however that the memory requirement of the fast mode has not be thoroughly considered. Most importantly, we would need to set aside additional memory *if* the image mode and fast mode observations will be done simultaneously.

The bottom line is this: We can live with 8 Mbytes of memory as designed, but it will put some constraints on the possibilities of combinations of observing modes. If the memory is relative inexpensive, from the point of view of this project, I strongly suggest that we have at least 12 Mbytes of memory.

VI.b. CPU Cycles Accounting

In this section, we give the rough accounting of the number and types of CPU operations needed in the stellar image analysis and the tracking algorithm discussed earlier. The major operations in the current implementation are **IF**, **IAAD** (integer add) and **IMUL** (integer multiply). The number of integer division is very small. We do not include considerations of the memory addressing here. These are **MOVE**s from memory to register and vice versa, as well as PUSH and POP from the stack. Since memory addressing can be quite subtle in high-level languages, there may be a lot of quicksands and elbow rooms in these unaccounted-for operations. Optimization and accounting for memory addressing is probably best done at low-level language with the specific CPU architecture in view. Here we discuss operation/CPU-cycle accounting in four major areas.

First, Table VI.1 gives an accounting of CPU cycles for the reference processing. This processing is not subject to the 10-s constraint. It appears that the most time-consuming part are 1) 1,000,000 *IF*s in scanning through the whole frame to locate the bright pixels and the bright stars and 2) the potentially large number of *IMUL*s in analyzing the bright stars.

TABLE VI.1

CPU Accounting for Reference Frame Processing

Operations	**IF**	**IADD**	**IMUL**
Scan Pixels above Threshold			
Per Pixel	1		
Total for a Frame	1,000,000		
Building Bright Stars			
Per Star	500	500	500
Total for 200 Bright Stars	100,000	100,000	100,000
Sorting Bright Stars Assuming 200 Bright Stars	1,600		
Filtering Good Guide Star			
Per Candidate Guide Star	30	50	50
Total for 50 Candidate Guide Stars	1,500	2,500	2,500

Second, Table VI.2 gives the accounting of the tracking algorithm in the 10-s frame processing. It appears from Table VI.2 that the analysis of the guide stars and the calculation of the drifts does not requires an inhibitive amount of CPU cycles. On a SPARCstation 1, these two steps combined take less than 1 second.

TABLE VI.2

CPU Accounting for Tracking

Operations	IF	IADD	IMUL
Building Guide Stars			
Operations per Star	400	300	250
Total for 10 Guide Stars	4,000	3,000	2,500
Drift Calculation	5	50	50

Third, we discuss the CPU-cycle accounting of the shift-and-add and the threshold mode processing. Whether these operations will be performed by a SAAHC or a dedicated microprocessor remains an open issue. Table VI.3 lists the minimum requirements for these processing.

TABLE VI.3

CPU Accounting for Shift-and-Add and Threshold Mode Processing

Operations	IF	IADD	IMUL
Scan Pixels above Threshold			
Per Star	1		
Total for a Frame	1,000,000		
Building Bright Stars			
Per Star	500	500	500
Total for 50 Bright Stars	25,000	25,000	25,000
Shift-and-Add			
Per Pixel		1	
Total		1,000,000	

Finally, we discuss the CPU-cycle accounting for the compression algorithm discussed in Section V. As in the shift-and-add processing, a mundane task appears to consume a major fraction of the CPU cycle. In the case of VBTWL compression, this task is *the evaluation of the most significant bit for a given integer.* Current implementation in C uses consecutive downshifting of the integer and check whether the downshifted integer is equal to zero. Table VI.4 lists a rough accounting of the number of operations. Since the major operations are **IF**s, we have not listed the accounting for **IADD** and **IMUL**. Furthermore, we are using C to manipulate the bits to construct the telemetry stream. The result is shown in the encoding section of Table VI.4. Here a block size of 16 by 16 is adopted. The compression with three predictor scheme plus encoding takes about 17 sec on a SPARCstation 1+. Improvement in the efficiency of compression efficiency can come from the following directions.

- First, it is conceivable that a more efficient algorithm can be build in machine code which reduces the usage of CPU cycles in the MSB and encoding procedures.

- If the first avenue is not fruitful, we can always reduce the number of predictors we use. The code should allow the turn-on and turn-off of certain predictors. The inclusion of the single-differencing predictor appears to offer only marginal improvement in compression ratio.

TABLE VI.4

CPU Accounting for Image Compression

Operations	**IF**
Optimization of Predictor/Tiering Structure	
Calculating the Most Significant Bit	
Per Predictor Per Twixel	~7
Total for 262144 Twixels Per Predictor	1,800,000
Total for 3 Predictors	5,500,000
Encoding	
Per Twixel	(4 – 10)
Total for 262144 Twixels	~ 2,000,000
Overhead	
Per 16 by 16 Block	30
Total for 1024 Blocks	30,000

VII. Threshold and Fast Mode Processing for The Blue Instrument

In this section, we discuss the processing algorithm of the threshold mode and fast mode for the blue instrument. Even though these algorithms have not be fully implemented, there appear to be no major difficulty in their design and implementation. Their memory and CPU-cycle requirements also appear within bounds. The greatest uncertainty is probably the compression of the fast mode data.

VII.a. Threshold-Mode Processing

The threshold mode observation uses spare telemetry bandwidth to yield useful scientific data. The purpose is to know in general what's going on within the FOV at a frame-by-frame basis. This mode should pick out N brightest pixels/images within the frame and downlink the information.

The threshold mode does not in itself cost much memory; it could, however, be an large user of CPU cycles. Overall, the threshold mode processing is very similar to the scanning and analysis for the bright stars discussed in Section II.f. See the Section VI.b. for an accounting of the CPU cycles for the bright star scanning procedure. Current design philosophy for the threshold mode processing is as follows, depending on whether we have enough CPU cycles.

- If we have spare CPU cycles, then we simply adopt the same algorithm as the bright star processing for each frame.

- If we do not have spare CPU cycles, then we need to consider ways to economize the threshold mode processing.

In the following, we discuss possible ways to reduce the processing requirement for threshold mode. Operationally, the threshold mode processing can be broken into two parts. The first part is the scanning and the second part is the analysis.

Scan for Bright Pixels

We need to scan through all of the frame pixels. If we check every pixel to see whether it is above the threshold or not (most will not be above the threshold); there will be altogether ~1,000,000 **IF** operations. However, it is conceivable that we can halve the number of **IF** operations.

We begin by considering the case when the threshold mode data is being processed contemporaneously with the image mode. It is most natural to embed the threshold pixel scanning in the shift-and-add as discussed in Section IV. Using the Approach 3 in Figure IV.3 as an example, after fetching the pixel count from the memory, the SAAHC (or microprocessor) will first check for the threshold. In the rare occasion of a bright pixel, an interrupt along with the address of the bright pixel will be sent to the CPU and initiate the analysis. In Figure IV.3, we show that the checking is applied for every pixels. We could, however, sum the counts for the two neighboring pixels (which is then added to the final image) and check the sum against

some threshold value. In other words, the two **IF**s in Figure IV.3 is combined into one. In this way, the number of **IF**s is effectively cut in half.

On a SPARCstation 1+, an **IF** is estimated to take 2μsec. The difference between processing 1,000,000 and 500,000 **IF**s is 1 whole second. Keeping in mind that a SPARCstation is not a realistic representation of the performance of the CPU, the 1-second difference does provide a critical example of the importance of savings in the number of **IF**s in the 10-s frame processing.

In the case when we carry out fast mode observation contemporaneously, a similar scanning procedure can be applied. Th only difference is that we do not perform the addition into the final image.

In principle, we can further reduce the number of **IF** operations by checking the sum of more than two consecutive pixels. My current feeling is that checking the sum of two in the threshold scanning should be sufficient.

Analysis

The nright pixel threshold should be set such that we are only processing a very small fraction of the pixels in the frame. There are two approaches.

- We can pick up only the information related to that particular bright pixel.
- We can perform some analysis to the image containing the bright pixels. This processing can be exactly the same as the bright star/guide star analysis.

My feeling right now is that the bright star processing discussed in Section II does not seem to require a very large amount of PCU power. I do not any compelling reason why we should not perform the analysis for each bright star. Of coruse, we will not subject the subject the bright stars to the good guide star test, since the guide stars will not be changed during the 1000-s integration.

VII.b. Fast-Mode Processing

The processing for the fast mode is more CPU intensive than memory intensive. As far as I can tell, no timing information is provided to the DPU for individual photons, other than in the engineering mode. We should rely on the DPU's own clock to provide the timing information. This is done by reading the accumulating frame which is being written into by the detector in real time. The procedure is sketched below.

1. Every Δt, the DPU system clock (assumed to have sufficient accuracy) pulse the CPU for an action.

2. The CPU reads a specified region in the accumulating frame and stores the output at a certain location. Let's refer to this array of data as a window stack.

3. The CPU calculates the difference between the current window stack and the window stack from the previous read. The result is the counts accumulated during this Δt. Let's refer to this data set as a window.

4. The final data is a sequence of windows which need to be compressed and formulated into a telemetry stream.

Issues for the algorithm design include:

- The processing in steps 2, 3 and 4 above must be finished in Δt.

- The limit in telemetry bandwidth and the brightness of the region of interest will constrain the size of the window and the timing resolution Δt.

- The fast mode, more than the other two mdoes, is driven by the science we wish to achieve. Possible targets of fast mode observation include: core of globular cluster with high time resolution and moderate spatial resolution (ms pulsar in GC), field of low count rate with high time resolution and moderate spatial resolution (field fast pulsar), field of high count rate with moderate time resolution and low spatial resolution (stellar seismology). The fast mode processing should be versatile enough to handle these situations.

- The compression algorithm appropriate for the fast mode data is most likely different from the compression of the final image. It should be able to handle the various situations arised from the wide range of scientific objectives of the fast mode.

More works are needed to develop the fast mode processing algorithm. However, I do not forsee major obstacles in the completion of the algorithm and achieving the scientific goals.

VIII. Glossary

Frame	A 1024 × 1024 array generated by a 10-s integration of the blue instrument.
Pixel	A resolved unit in the 1024 × 1024 frame, =(0.5 arcsec)^2
Reference frame	The frame taken before the 100-frame integration. Information extracted from this frame will be used for tracking.
Working frame	A frame which is obtained after a 10-s integration and is currently being processed by the DPU for tracking, shift-and-add, threshold and fast mode processing.
Accumulating frame	A frame which is being directly written to by the increment processor, concurrent to the processing of the working frame by the DPU.
Final image	A 512×512 array resulted from the summation of 100 frames.
Twixel	A resolved unit in the final image, = 2 × 2 pixels = 1(arcsec)^2.
Bright stars	Stars which contain pixels with counts above a certain threshold and to whom the image analysis algorithm are applied.
Goodness	A set of values reflecting the the quality of the stellar image base on a certain set of disgnostic criterions.
Guide stars	Bright stars with good image qualities (goodness of zero) which are used for tracking throughout the 100-frame integration.
Drift	The translation of the aspect of the working frame with respect to the reference frame. Δx and Δy.
Roll	The relative orientation angle between the working frame and to the reference frame, measured with respect to a pre-defined center. Θ.
SAAHC	The Shift-And-Add hardware Coprocessor. This divice external to the microprocessor is responsible for the efficient shifting-and-adding of the columns of frame data to the final image.
VBTWL	The Variable-Block-Tiered-Word-Length scheme for image data compression.
Predictor	The quantity used in the decorrelation procedure of the compression scheme to remove the predictable component of the information.

DPU PROCESSING FOR XMM/OM

TRACKING AND COMPRESSION ALGORITHM

DOCUMENT 3

XMM-OM/PENN/TC/0004.03

Author: Cheng Ho

CH6/17/1991

This release of the document "Tracking and Compression Algorithm" (XMM-OM/PENN/TC/0004.03) is to provide an updated discussion on the tracking algorithm. Only chapter 3 of the document is being updated. Chapter 3 of the previous release including figures (XMM-OM/PENN/TC/0004.02) is superceded by this release in its entirety. Changes in this release of Chapter 3 are:

• The original section III.d on Image Quality and related figures are removed. A full discussion on image quality will be presented in a separate document.

• The tracking algorithm has undergone a major overhaul. We have now included the statistical weighting of individual guide stars. The statistical weighting discussion is inserted as III.c. The discussion of the tracking accuracy in III.d is revised accordingly. Furthermore, we have developed two different schemes of handling outlier stars. These are extensively discussed in III.e. These two new developments (statistical weighting and outlier handling) enhance the accuracy of the tracking and also increase the robustness of the algorithm under unfavorable conditions.

III. Tracking

III.a. Basic Formalism

Suppose we have located and chosen N good guide stars within the field of view in the reference frame at (u_i, v_i) where i runs from 1 to N. At a later moment, we take an integration of a frame and determine the new locations of the guide stars at (x_i, y_i). Our task is to determine the change in the aspect of the current frame with respect to the reference frame, i.e. to calculate the drift in the x and y direction as well as the roll with respect to a particular roll center taken to be the origin. The selection of the roll center is *arbitrary*. However, see discussions below on a natural default choice of the roll center where the errors of the calculated drifts are *formally* minimized.

Applying a drift and roll $(\Delta x, \Delta y, \Theta)$ to the guide stars, their new locations are given by

$$\hat{x}_i = u_i C - v_i S + \Delta x \qquad (III.1a)$$

$$\hat{y}_i = u_i S + v_i C + \Delta y, \qquad (III.1b)$$

where $C = \cos\Theta$ and $S = \sin\Theta$. (Without causing any confusion, we use C here for the cosine of the roll angle, while in the rest of this document, C refers to the total count in a stellar image. In III.b., we shall see that the cosine equals to unity for all practical purposes.) With the observed (x_i, y_i), the drifts are calculated by minimizing

$$D^2 = \sum_i w_i D_i^2 \qquad (III.2)$$

where

$$D_i \equiv (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2, \qquad (III.2a)$$

ans w_i is the statistical weight for the i-th guide star. We shall discuss the assignment of w_i later. For the moment, let's assume that the weight w_i is independent of the drift and roll (Δx, Δy, Θ). Differentiating D^2 with respect to Δx, Δy and Θ at the minimum yields

$$\Delta x = (X - UC + VS)/W \qquad (III.3a)$$

$$\Delta y = (Y - VC - US)/W \qquad (III.3b)$$

$$SP + CQ = 0, \qquad (III.3c)$$

where

$$U = \sum_i w_i u_i, \qquad (III.4a)$$

$$V = \sum_i w_i v_i, \qquad (III.4b)$$

$$X = \sum_i w_i x_i, \qquad (III.4c)$$

$$Y = \sum_i w_i y_i, \qquad (III.4d)$$

$$W = \sum_i w_i. \qquad (III.4e)$$

$$P = \left[\sum_i w_i(x_i u_i + y_i v_i)\right] - (XU + YV)/W, \qquad (III.4f)$$

$$Q = \left[\sum_i w_i(x_i v_i - y_i u_i)\right] - (XV - YU)/W. \qquad (III.4g)$$

For our application, the roll angle is always small and P is always greater than zero, thus the roll angle is given by

$$\Theta = \sin^{-1}(-Q/R) = \sin^{-1}\left[-Q/(P^2 + Q^2)^{1/2}\right]. \qquad (III.5)$$

III.b. Small Roll Angle Approximation

As long as Θ is small, we can make the following approximation in calculating the roll angle,

$$C = 1, \quad \text{and} \quad S = \Theta = -Q/R, \qquad (III.6)$$

where

$$R = (Q^2 + P^2)^{-1/2} \simeq P + Q^2/2P. \qquad (III.7)$$

In this small roll angle approximation, calculation of Θ via equation (III.7) does not need the square root operation as in equation (III.5). This approximation will introduce error which is of order Θ^2. Since the roll angle is expected to be on the order of 0.002 radians from the beginning to the end of a 1000-s integration, using equation (III.7) will not introduce any significant error (see below for estimate in the error of Θ).

In summary, the drift and the roll of the satellite are calculated through the equations (III.4) which yields the quantities U, V, X, Y, W, P, and Q. From these quantities, we can calculate the drift and roll in the small angle limit:

$$S = \Theta = -PQ/(P^2 + Q^2/2), \qquad (III.8a)$$

$$\Delta x = (X - U + VS)/W, \qquad (III.8b)$$

$$\Delta y = (Y - V - US)/W. \qquad (III.8c)$$

III.c. Assignment of Statistical Weight

We need to specify the statistical weight in order to carry out the calculation of the drift and roll. The simplest form of the statistical weight is to set w_i uniformly to unity (*uniform weighting*). The calculation is much simplified and this has been

the adopted approach in earlier versions of the tracking algorithm. In this case, the uncertainty of the drift and roll is dominated by the worst tracking guide stars, usually the faintest one. Under nominal conditions, there might be a factor of 2 to 3 difference between the brightest and the faintest guide stars. The resulting accuracy is generally acceptable. This could, however, become a problem in a pathological field where the available guide stars are limited and they vary dramatically in brightness. For example, in one particular simulation of a sparse field through a narrow band (1%) filter, the brightest (320 counts/frame) and the faintest (20 counts/s) differ in brightness by a factor of about 16. A uniform statistical weight will then seriously affect the tracking performance. And it is especially in such cases that the tracking accuracy becomes important.

With some increase in processing, we can take into account the varying statistical weights of different guide stars. As discussed in Chapter II of this document, the guide star location algorithm approximately achieve the accuracy of $\sigma_{PSF}/C^{1/2}$ for the centroid of the star, where σ_{PSF} is the standard variation of the detector/system PSF. Thus the statistical weight of individual guide star locations is directly proportional to the counts in the image. Thus, we assign

$$w_i = C_i, \qquad (III.9)$$

where C_i is the count of the guide star image. This quantity is readily available from the guide star location algorithm. For a non-robust (no outlier data point biasing), this is an adequate and simple designation of the statistical weight for the guide stars.

In the actual implementation, we have two choices of incorporating the statistical weight. The first one is to use the counts obtained from the reference frame and apply it throughout (*reference count weighting*). The advantage of this approach is that U, V, W can be pre-calculated before the frame integrations. Some saving in processing could be gained. The disadvantages is that the fluctuation in U, V, W will be frozen. Such fluctuations would be carried over to the frame-by-frame tracking and they could lead to a finite offset in the tracking performance. The second approach is to use the counts for the guide stars in the current frame as the statistical weight (*current count weighting*). While this would be more favorable from the statistical point of view, additional processing would be required.

At the moment, the current count weighting scheme is adopted. CPU timing benchmark is needed to check whether the drift and roll calculation impose a severe processing penalty. If this turn out to be the case, then we would have to consider the reference count weighting scheme.

Figure III.1 shows the tracking result of a nominal situation. Fields with an average stellar density are simulated in white light with random walk drift. The upper left panel shows the distribution of the brightest stars. The upper right panel shows the locations of the guide stars selected by the guide star selection algorithm discussed in Chapter II. The middle left panel shows the deviation between the

calculated drift and the (simulated) real drift. The middle right panel shows the same result, only on a finer scale. The lower left panel shows the deviation of the calculated roll from the actual roll. The lower right panel shows the deviation of the observed (from guide star location) guide star locations to the projected (by applying the calculated drift/roll to the reference) guide star locations. In general, the more concentrated the cluster, the better the tracking performance. See discussions below on the application of these quantities for robust outlier handling. Figure III.2 shows the same case as in Figure III.2 except that the stellar density is taken to be in the galactic plane. Figure III.3 shows the case of the galactic pole.

III.d. Precision of the Tracking

Since the observed location of the guide stars (x_i, y_i) as well as the reference locations (u_i, v_i) will no doubt have uncertainties, we expect the calculated drifts and roll to have some uncertainty. We now estimate the errorbar of the tracking quantities and the calculated drift and roll in the case of using the count weighting scheme. We take the error bar for the stellar location in the x and y to be $\delta x_i \simeq \sigma_{PSF}/C_i^{1/2}$. It is straightforward to estimate the standard deviations for X, Y, U, and V:

$$\delta X = \delta Y = \delta U = \delta V = W\langle \delta x \rangle \simeq \sigma_{PSF} W^{1/2}, \qquad (III.10)$$

where $\langle \delta x \rangle = [\sum_i w_i^2 (\delta x_i)^2]^{1/2}/W$. Even though the values of $\delta X(\delta Y)$ and $\delta U(\delta V)$ are the same, we need to make a distinction between them. The quantities U and V are partly universal for the frame-by-frame tracking; errors associated with a linear dependence of them can be easily removed as a systematic error by an after-the-fact calibration to align the star positions with some absolute reference frame; this error is partially reflected in the finite average offset in the tracking performance. linear contributions of U and V in equations (III.8*b*) and (III.8*c*) does not affect the precision of the tracking.

The error of Q is

$$\delta Q \simeq \sqrt{R/W}\sigma_{PSF}, \qquad (III.11)$$

and the error of Θ is

$$\delta\Theta \simeq \frac{\delta Q}{R} \simeq \frac{\sigma_{PSF}}{\sqrt{WR}}. \qquad (III.12)$$

The standard deviations for the drifts are

$$\delta(\Delta x) \simeq \frac{\sigma_{PSF}}{\sqrt{W}} \left[1 + \frac{V^2}{WR}\right]^{1/2}, \qquad (III.13a)$$

and

$$\delta(\Delta y) \simeq \frac{\sigma_{PSF}}{\sqrt{W}} \left[1 + \frac{U^2}{WR}\right]^{1/2}. \qquad (III.13b)$$

As long as the roll center is chosen to lie within the field of view, both V^2/WR and U^2/WR will be less than unity.

Mathematically, applying a rotation and translation parameterized by the calculated drift and roll $(\Delta x \pm \delta(\Delta x), \Delta y \pm \delta(\Delta y), \Theta \pm \delta\Theta)$ to any particular point (x, y) point will yield the same result $(x' \pm \delta x', y' \pm y')$, *independent to the choice of the origin (roll center).* However, for efficiency reason, we will only perform on-board compensation to the translation (shift-and-add) to the satellite drift; no compensation will be applied for the roll. The shift-and-add procedure leads to the smallest errorbar at the roll center and the error is greater for points at greater distances from the roll center. Figure III.1 shows the deviation of the calculated drift from the real (simulated) drift. Figure III.2 shows deviation of the calculated roll from the real (simulated) roll. The spread of the deviations is consistent with the estimates given in equations (III.12) and (III.13).

We now discuss a special case where the errorbars of Δx and Δy are *formally* minimized. This is achieved by minimizing U^2 and V^2, i.e., choosing the center-of-mass of the guide stars as the roll center and setting $U = 0$ and $V = 0$. This simplifies equations (8*b*) and (8*c*) to $\Delta x = X/W$ and $\Delta y = Y/W$. The center-of-mass is the natural default roll center. However, with a moderate increase in arithmetics, we shall implementing the tracking algorithm as equation (8) which does leave open the possibility of choosing the roll center at *any* location.

In the simulation for Figure III.1, 10 guide stars are used with an averaged count rate of about 3500 counts/frame, i.e. $W \simeq 35{,}000$. The estimated (1-sigma) errorbar for $\delta(\Delta x)$ is approximately 0.003 arcsec (with σ_{PSF} of 0.5 arcsec). Visual inspection shows that the calculated fluctuation is slightly worse than this estimate. This is probably the result of finite contribution from the background which is essentially ignored in the discussions above. Similar conclusions is obtained for the cases shown in Figures III.2 and III.3.

III.e. Outlier Handling

In estimating the precision of the tracking calculation [Eqs. (III.12) and (III.13)], we have made the implicit assumption that all guide stars are well-behaved in that they are singular and does not vary with time other than statistical fluctuation. There are, however, situations where this assumption is violated. Such situations include:

- A bad guide star was chosen as a good one in the reference frame due to statistical fluctuation. For example, a binary image may disguise as a good guide star if the fainter stars is particularly weak and the brighter star is particularly strong in the reference frame. Nature, not statistics, could also play cruel trick on us by placing bright yet intrinsically variable stars in our frame. Even with the multiple guide star selection criteria discussed in Chapter II, there is no assurance that only good guide stars will be selected.

- Not enough good guide stars are located in the reference frame. We are forced to use some bad guide stars. Note that we need at least two guide stars to calculate the drift and roll.

• The detector's nonlinearity/nonuniformity could also offset the guide stars' location, even when the guide star itself is well behaved all the way. For example, such distortion could occur at the boundary of the MCP hexagonal pattern.

We have examine two methods of handling the outliers. The first one is detect the outliers and exclude them outright. The second one is to assign smaller weight to the outlier by modifying the weighting factor.

III.e.1. Outlier Exclusion

We begin by examining the guide star consistency through the calculated deviation from equations (III.1) and (III.2). If some of the guide stars turn sour in the tracking frame or a bad guide star was chosen in the reference frame, then these star will have large deviations, i.e., inconsistent with the calculated drift which is dominated by the rest of the presumably well-behaved guide stars. Once we single out the outliers, then excluding it from the drift calculation will enhance the precision of the tracking.

The outlier exclusion algorithm is implemented as follows.

1. We calculate the drift and roll with the current set of guide stars using equations (III.4) to (III.8).

2. We use the calculated drift and roll to calculate the deviation of the observed from the expected locations for individual guide stars. This is done by applying Δx, Δy and Θ (in the small roll angle limit) in equation (III.1) and calculating the deviation for individual guide star D_i^2 defined in equation (III.2).

3. The deviations D_i^2 are sorted by magnitude. The largest deviation $D_{i,\max}^2$ is examined for two outlier exclusion criterions:

$$D_{thld} \text{ exclusion :} \qquad D_{i,\max}^2 > D_{thld}^2, \tag{III.14}$$

or

$$\Omega \text{ exclusion :} \qquad D_{i,\max}^2 > \Omega \times (D^2 - D_{i,\max}^2)/(N_{ggs} - 1), \tag{III.15}$$

where D_{thld}^2 and $\Omega = \omega^2$ are two predetermined parameters for the drift calculation. If either equation (III.14) or (III.15) is satisfied for the outermost guide star, then we've found an outlier and it is excluded from the current guide star list. In each iteration, we only exclude one outlier.

4. With the updated current guide star list, we repeat step 1. The iteration is terminated if no guide star is excluded. Furthermore, to avoid a run-away process where the procedure does not converge, we impose the following additional exit conditions: *a*) the number of excluded guide star should not be greater than a certain number $N_{max,outlier}$, *b*) the number of used guide star should not be less than $N_{min,gs}$. Currently, we use $N_{max,outlier} = 3$ and $N_{min,gs} = 4$.

The key step in the outlier exclusion is equations (III.14) and (III.15). We now discuss the implications of this procedure. The objective of the D_{thld} exclusion is

straightforward. The selected D_{thld} is basically an estimate of the largest possible acceptable deviation due to statistics. The theoretical basis of the intuitive Ω exclusion, on the other hand, is more convoluted.

Suppose one of the guide stars turns bad in the current frame, then the Ω exclusion is roughly equivalent to excluding a star which has a deviation such that

$$D_i/\langle D\rangle > N_{ggs}\omega/(N_{ggs}-\omega), \qquad (III.16)$$

where $\langle D\rangle$ is the average deviation we expect from the *good* guide stars. In arriving at equation (III.16) we have taken into account the influence of the bad guide star to the current drift calculation. Several key features in equation (III.16) need to be noted. First, the exclusion is not a linear function of ω; it approximates linear function only when $N_{ggs} \gg \omega$. Second, if $\omega \gtrsim N_{ggs}$, then equation (III.18) cannot be satisfied; the Ω exclusion does not pick up any outlier. In the current implementation, we use $\Omega = 16$. If $N_{ggs} = 10$, then the Ω exclusion is equivalent to a 6-sigma exclusion, if we think of $\langle D\rangle$ as the standard deviation.

In summary, there are four adjustable parameters in the outlier exclusion scheme, D^2_{thld}, Ω, $N_{max,outlier}$ and $N_{min,gs}$. These four numbers should be fine-tuned by extensive simulations and more importantly, applications to realistic data and detector characteristics.

III.e.2. Robust Estimation

The use of robust estimator is well developed. *Numerical Recipe* contains a discussion on the basics of these methods. Here we simply discuss a twist to the robust estimator as in the current implementation.

The idea of robust estimator is to modifying the statistical weighting such that outlier data point will give proportionately smaller contribution. In our current implementation, we modify the statistical weight as (cf. Eq. III.9])

$$W'_i = C_i/(1+\rho C_i D_i^2), \qquad (III.19)$$

where D_i^2 is the deviation calculated through equation (III.2) as discussed in the outlier exclusion algorithm, and ρ is an externally specified parameter.

The essence of the modifier $M_i = (1+\rho C_i D_i^2)^{-1}$ is to simulate the *Lorentzian* distribution. See page 541 of Numerical Recipe for more discussions. Formally, if we assume the underlying distribution is Lorentzian, rather than Gaussian, then the drift and roll are calculated through a set of simultaneous non-linear equation similar to equation (III.4) except that the summations will now include the M_i factor. Since D_i depends on the calculated drift and roll, it is a non-trivial task to find the solution.

In the implementation, we apply an iterative scheme where the calculated drift and roll is fed back into the calculation of the statistical weighting [Eq. (III.18)] in the subsequent iteration. We start the iteration with the non-robust estimation,

i.e., setting the weight modifier to unity for the initial iteration. The iteration is terminated after a maximum number of iterations or the improvement in the averaged deviation (D^2/W) becomes negligible.

The meaning of the modifier parameter ρ is the following: Stars whose deviation follows

$$D_i^2 \gtrsim 1/\rho C_i \qquad (III.20)$$

will have a significantly suppressed statistical weight. Compared to the estimate of star location fluctuation $\sigma_{PSF}/C_i^{1/2}$, this means that stars whose deviation is greater than $\sigma_{PSF}/\rho^{1/2}$ standard deviation will be suppressed. In other words, in order to achieve suppression of data points which are k-standard deviation away from expectation, we need to specify ρ as σ_{PSF}/k^2. Currently, we adopt $\rho = 0.05\sigma_{PSF}$ which corresponds to a suppression of point beyond ~ 4.5 standard deviation.

To conclude, we have developed two schemes to handle the possible presence of outliers. The outlier exclusion algorithm applies an abrupt exclusion to eliminate outliers (it's either *in* or *out*), while the robust estimation uses more gradual suppression. The requirement on real-time CPU cycles for these two algorithms appears to be comparable. One can probably identify situations where one algorithm could work better than the other and vice versa. Overall, the robust algorithm has the advantage of having fewer major parameters (ρ versus D_{thld} and Ω), the meaning of this parameter is also more transparent. It is also easier for parameter optimization as the Optical Monitor continues to evolve in the future.

III.f. Summary

In summary, a paradigm has been developed to calculate the satellite drift and roll. There are multiple variations within this paradigm: We can choose among three different weighting factors and we have a choice between two different kinds of outlier handling scheme (plus no outlier handling at all). The decision of which algorithm to use depends on two factors.

First, the algorithm must be efficient enough that the CPU can carry out the calculation without stress. CPU benchmarking/timing is important once the appropriate hardware becomes accessible. Given sufficient CPU power, the current top contender for the tracking algorithm configuration is the iterative robust estimation with current count weighting.

Second, the tracking algorithm must be able to yield acceptable performance over a wide range star fields. The results of tracking with pathological star field is given in a separate document (XMM-OM/PENN/TC/0019.xx). We expect both this document and the compilation of tracking for pathological fields to evolve over the course of this project.

Figure Captions for Chapter III

Figure III.1 — Tracking performance for an average star field in white light. The upper left panel shows the distribution of the stars. The upper right panel shows the locations of the guide stars selected by the guide star selection algorithm discussed in Chapter II. The middle left panel shows the deviation between the calculated drift and the (simulated) real drift. The middle right panel shows the same result, only on a finer scale. The lower left panel shows the deviation of the calculated roll from the actual roll. The lower right panel shows the deviation of the observed (from guide star location) guide star locations to the projected (by applying the calculated drift/roll to the reference) guide star locations. In general, the more concentrated the cluster, the better the tracking performance. See discussion in III.e on the application of these quantities for robust outlier handling.

Figure III.2 — Same as Figure 1 with a simulated dense (galactic plane) stellar distribution.

Figure III.3 — Same as Figure 1 with a simulated sparse (galactic pole) stellar distribution.

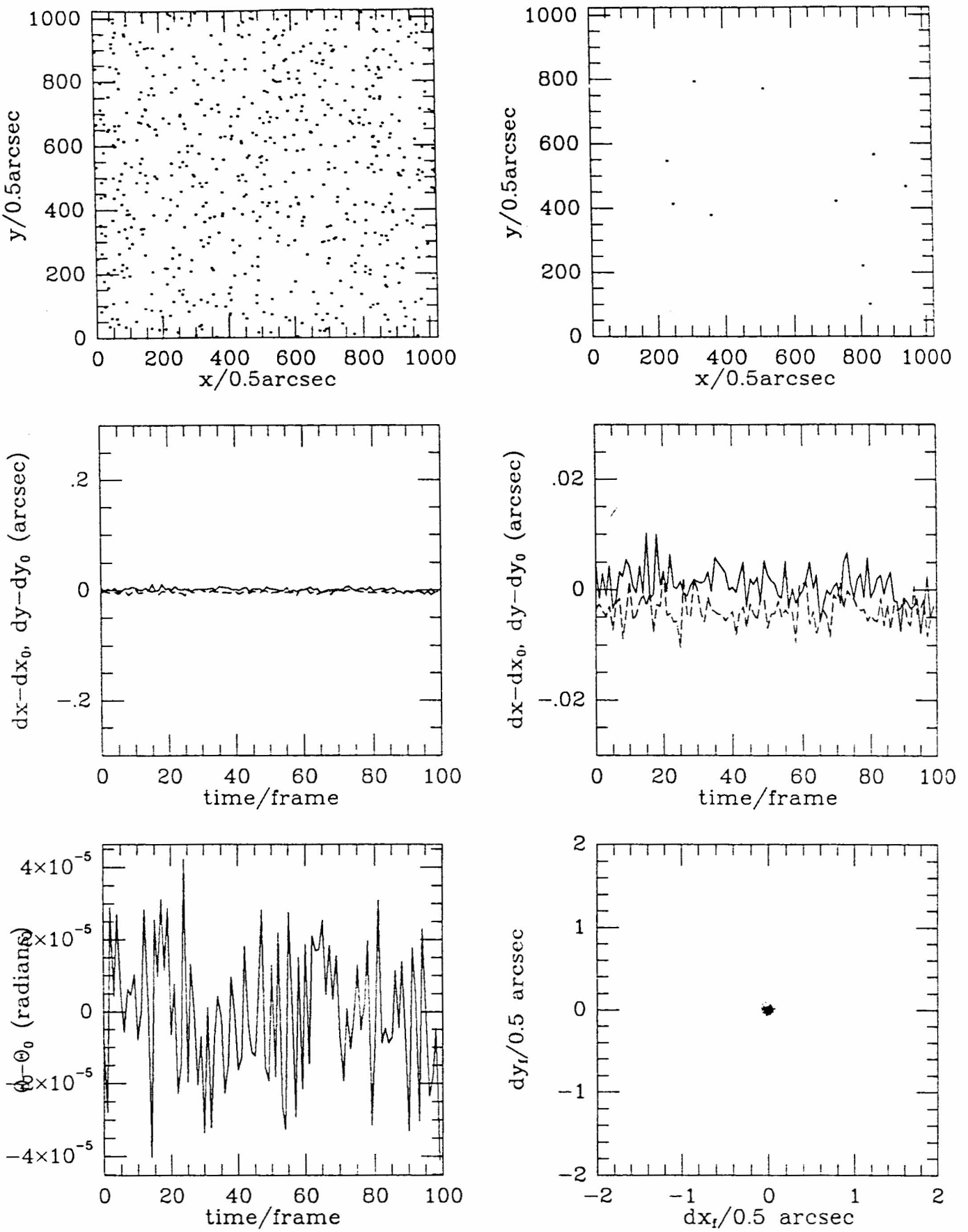

y/0.5arcsec
x/0.5arcsec
dx−dx₀, dy−dy₀ (arcsec)
time/frame
dx−dx₀, dy−dy₀ (arcsec)
time/frame
Θ−Θ₀ (radians)
time/frame
dy_I/0.5 arcsec
dx_I/0.5 arcsec

Fig. III.1

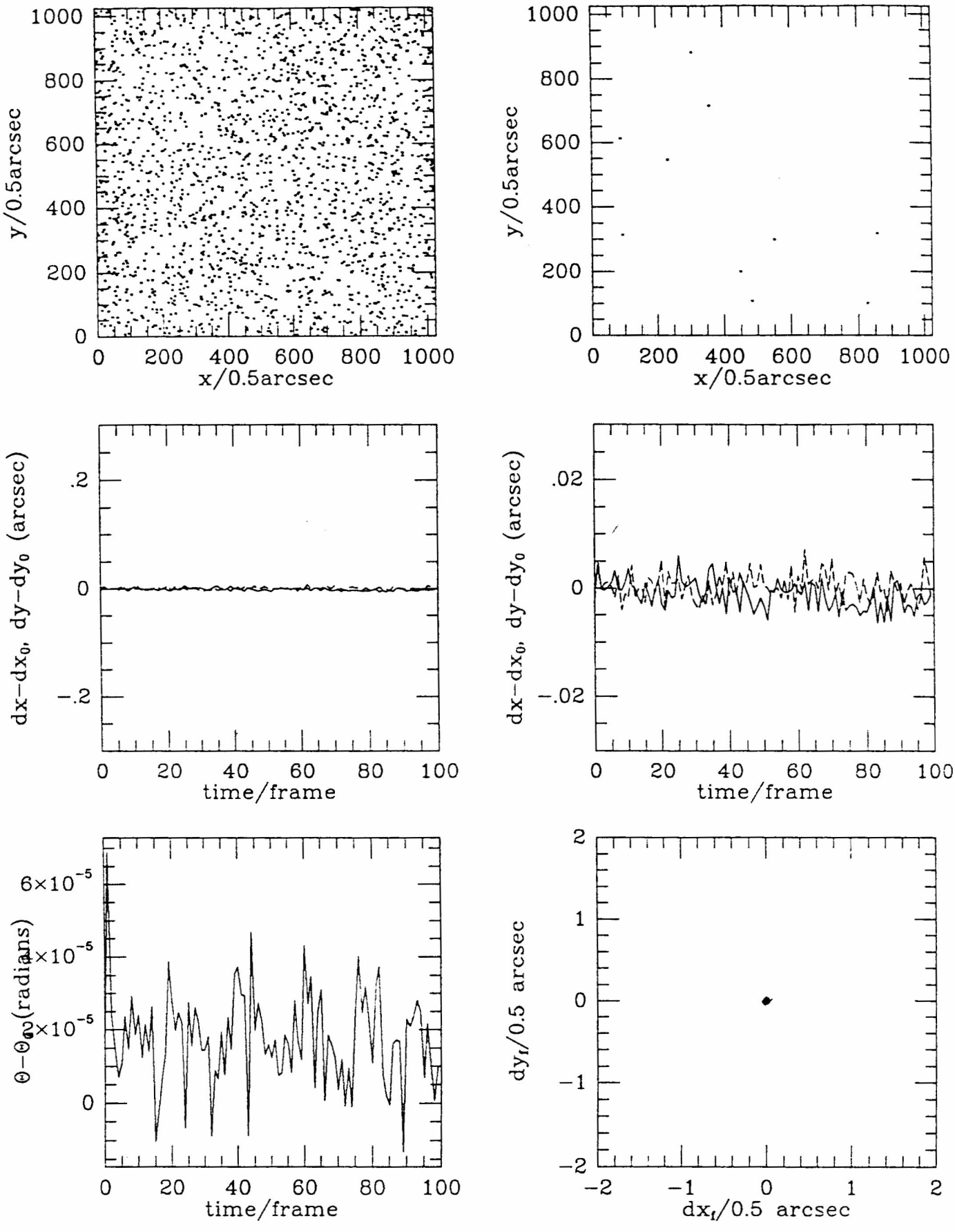

y/0.5arcsec
x/0.5arcsec
dx−dx0, dy−dy0 (arcsec)
time/frame
Θ−Θ0(radians)
dyf/0.5 arcsec
dxf/0.5 arcsec

Fig. III.2

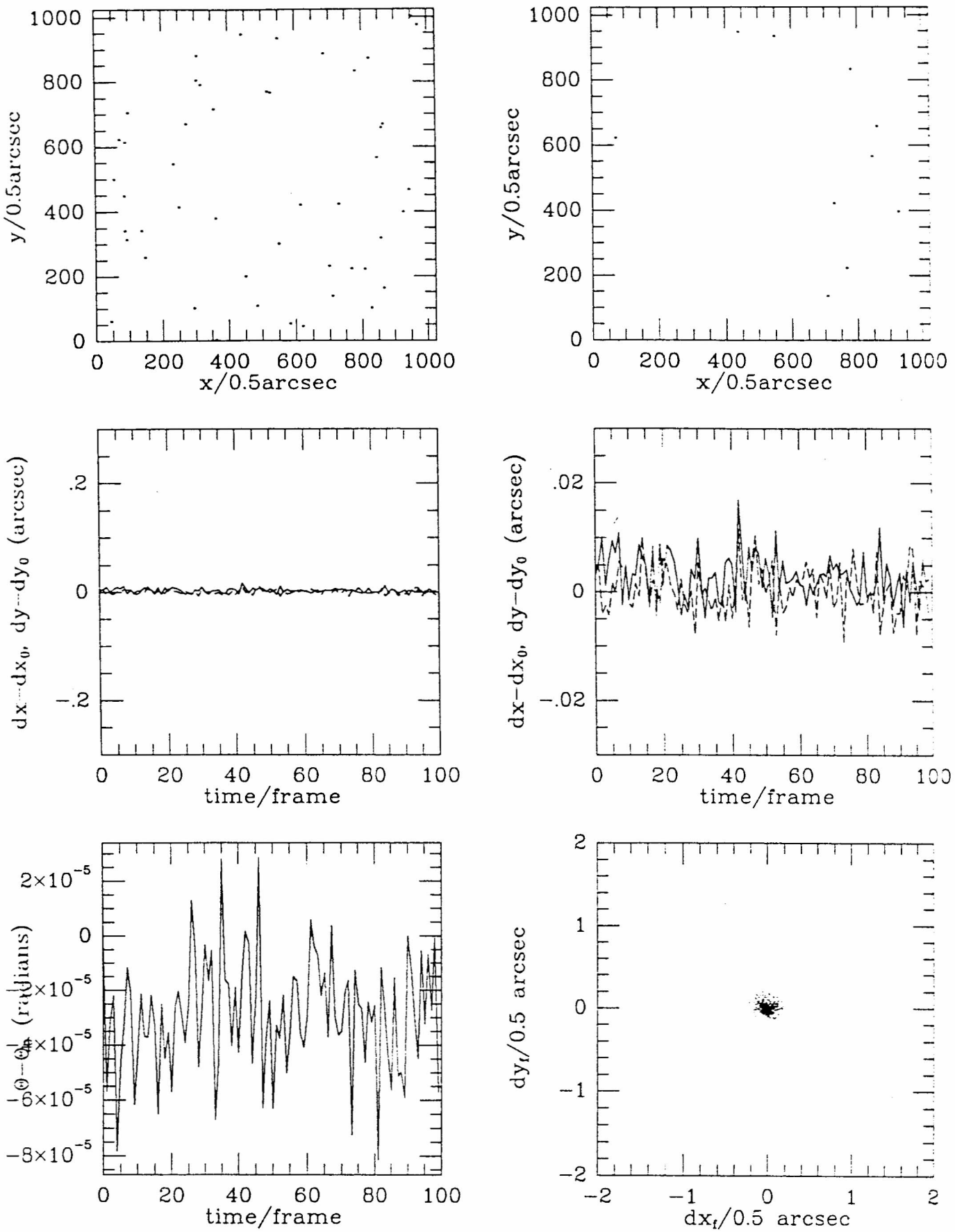

y/0.5arcsec
x/0.5arcsec
dx−dx₀, dy−dy₀ (arcsec)
time/frame
dx−dx₀, dy−dy₀ (arcsec)
time/frame
Ω−Ω_ave (radians)
time/frame
dy_f/0.5 arcsec
dx_f/0.5 arcsec

Fig. III.3