Cross Correlation and IROS Imaging methods for a static Scanning Sky Monitor camera

Ravi Shankar BT & Dipankar Bhattacharya

Jan, 2002

Abstract

This report describes the procedure followed to develop the reconstruction algorithm for the Scanning Sky Monitor. It also gives an account of the software developed so far in this regard and describes the logic used in them. The software written for implementing the simulation of the camera, and the algorithm of Iterative Removal Of Sources have also been explained.

Contents

Chapter 1

INTRODUCTION

1.1 One-Dimensional Camera

1.1.1 Simulation

To begin with, here is a description as to how the simulation is carried out. We started off with a simple box-type camera, with a single mask pattern (out of six available patterns). The camera will simply be a box, with a mask pattern at one end, and a plane detector at the other. The mask consists of 63 closed and open mask elements, each of the same width. Several open or closed mask elements may however be placed next to each other causing the length of the open and closed areas to vary in integral multiples of basic mask element width. The mask elements in the mask patterns have been represented as series of 1's and 0's. [1 represents an open mask element, and 0 a closed one]. Figure 1.1 shows the six different mask patterns that have been designed.

To simulate the photon strike, we employed the Monte Carlo methods. So Park Miller Minimal standard algorithm, with shuffling, was used for generating random numbers. These random numbers are used to get the positions of the photon strike on the mask plate. It is checked if the positions coincide with an open slit, only then will the photons be allowed to strike the detector. For a given angle of the source, the position of the photon-fall on the detector can be calculated. In the real situation, there will be an uncertainty in reading out the position of photon-strike on the detector and follows a Gaussian distribution. To simulate this aspect, for every position of photon-strike(on detector) calculated, a Gaussian of finite σ(a few mm) is added to it. The value of the σ is one of the parameters to the simulation code [and a value of 0.5 mm has been used, along the mask coding direction, the X-direction]. The effect of this is that, the photons get smeared across the detector plane. Figure 1.2 shows the comparison between the shadow patterns of photons on an ideal detector, and that on a detector with finite resolution.

1.1.2 Image Reconstruction

The image reconstruction procedure starts at this point. The concept of imaging software developed has been adopted from the PhD thesis *A Coded-Mask Imager As Monitor Of Galactic X-Ray Sources* by Johannes Joseph Marie in 't Zand, Utrecht University, 1992.

The detector is assumed to be made of bins. And the number of bins was selected to be equal to that of the number of mask elements, that is 63. The sky is modelled as discrete sky elements. From one sky element to the next, the shadow of the mask pattern produced by a source at that sky element moves by one detector bin. Given 63 each of mask elements and detector bins, 126 sky elements can be realised along the X-direction.

The angle corresponding to a sky element k is given by,

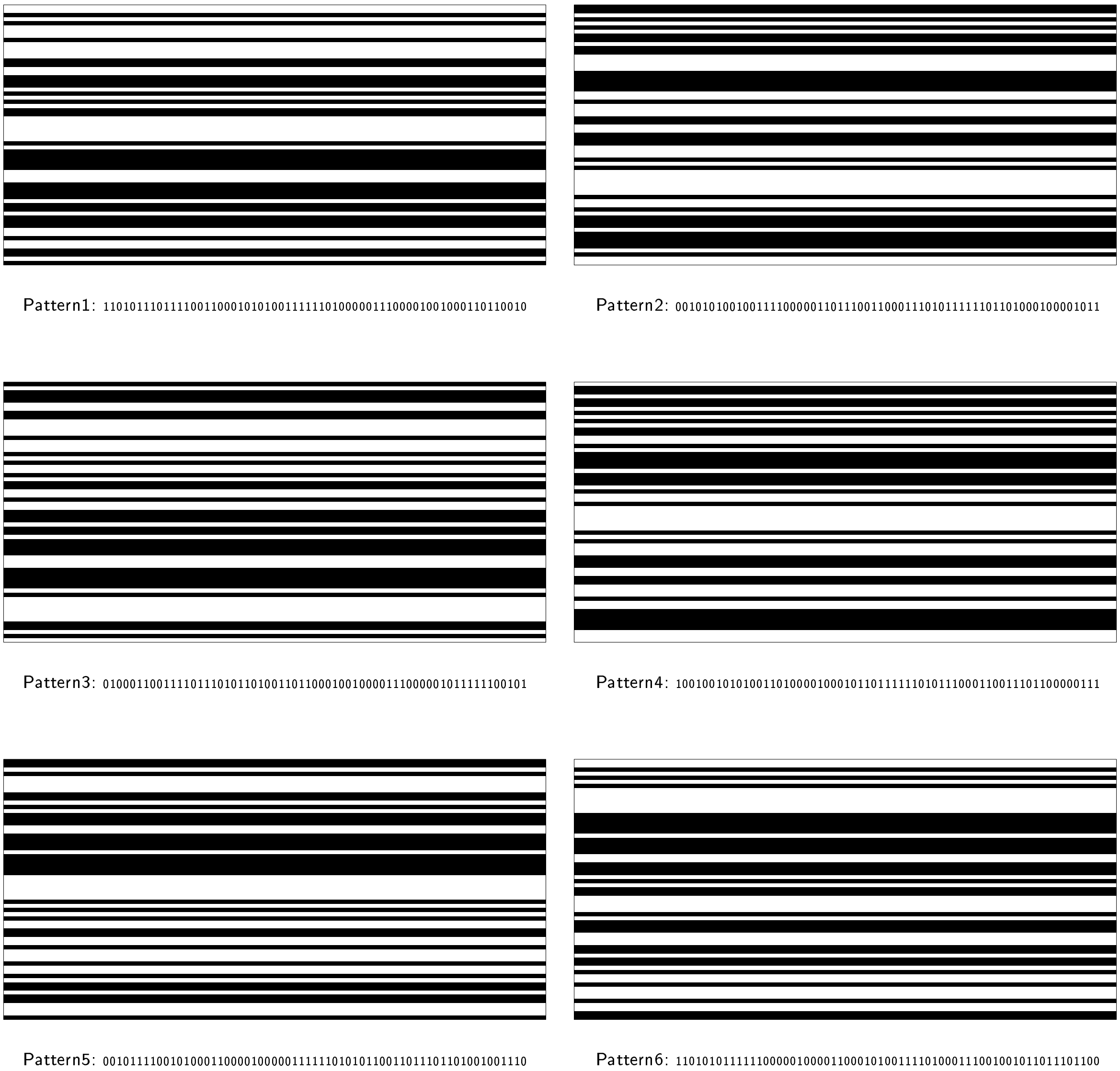


Figure 1.1: *The six Mask Patterns and the corresponding 63 mask elements in 1's and 0's. The first mask element is the one in the top in each of them.*

$$\theta_x(k) = tan^{-1}\left(\frac{62\ -\ k}{Height} * PerDetectorBinWidth\right) \quad \forall\ 0 \leq k \leq 62$$
$$\theta_x(k) = tan^{-1}\left(\frac{k\ -\ 63}{Height} * PerDetectorBinWidth\right) \quad \forall\ 63 \leq k \leq 125$$

where,
$Height = 300\ mm$
$PerDetectorBinWidth = \left(\frac{60}{63}\right)\ mm$

A source at some sky element k casts a shadow of a portion of the mask on the detector. This portion and the location of the shadow on the detector depends on the sky element k. From the simulated photon strikes on the detector, one constructs a count of photons, in each detector bin. Ideally, the illuminated bins should each receive the same number of photons, but the random nature of photon strikes causes a variation of counts which we call "Poisson Noise". In addition to the source, the detector also receives photon counts due to the isotropic sky background and homogeneous detector background. To include this in the simulations, Gaussian deviates are added to the detector bin counts,

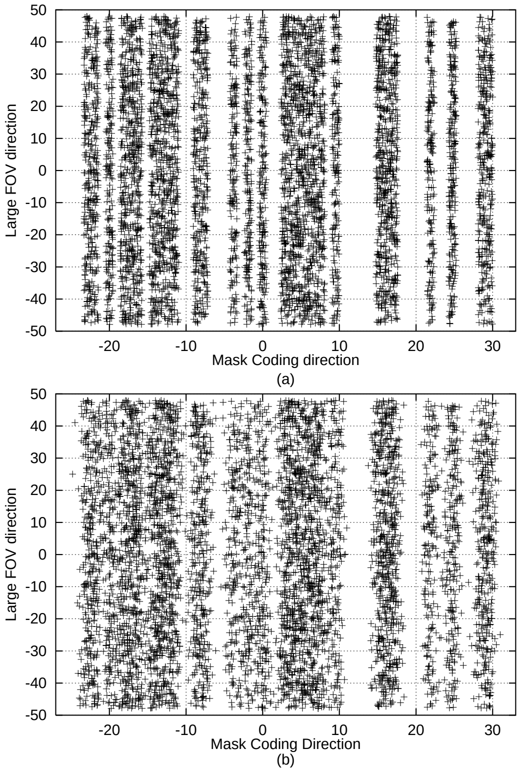


Figure 1.2: *Shadow Pattern (a) On an ideal Detector, (b) On a detector with a finite resolution, with a Gaussian of σ, 0.5 mm*

after scaling them to the desired noise level. Figure 1.3 shows shadow patterns in terms of photon counts in the detector bins. These patterns correspond to three different cases, an ideal detector, a detector with finite resolution, and a detector with finite resolution, with the background noise added to the bin counts, respectively.

We describe next the cross correlation method employed for reconstruction of the sky image. We start with the following definitions:

Detector Arrays

Using the count of photons in each detector bin, the detector array, d_j^k, corresponding to every sky element is generated.

The procedure is as follows:

Let the detector bin counts be, $\{b_0,\ b_1, \ldots,\ b_{62}\}$, and let the number of mask elements be represented as $N(=63)$; then corresponding to each sky element k, following detector vectors D^k can be listed:

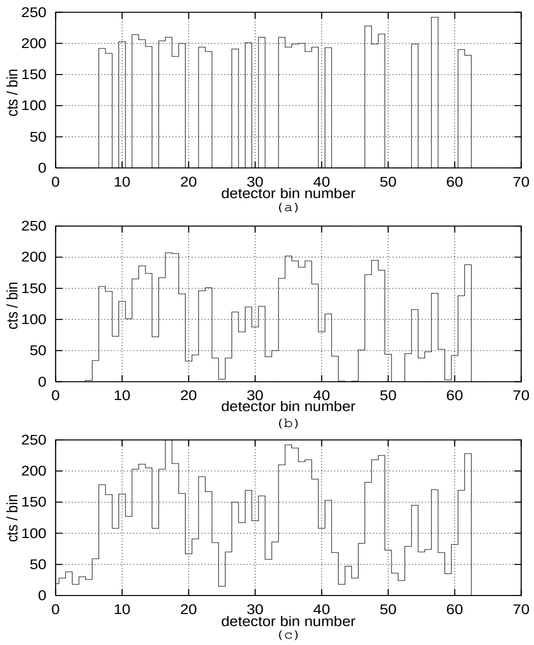


Figure 1.3: *Shadow patterns, in terms of photon counts per detector bin, (a) for an ideal detector, (b) for a detector with finite resolution, with a Gaussian of 0.5 mm and (c) for the detector in (b) with a noise of RMS value 10 cts/me*

$$
\begin{aligned}
D^0 &= \{b_{62}\},\\
D^1 &= \{b_{61},\ b_{62}\},\\
D^2 &= \{b_{60},\ b_{61},\ b_{62}\},\\
&\vdots\\
D^{61} &= \{b_1, \ldots\ldots,\ b_{60},\ b_{61},\ b_{62}\},\\
D^{62} &= \{b_0, \ldots\ldots\ldots,\ b_{60},\ b_{61},\ b_{62}\},\\
D^{63} &= \{b_0, \ldots\ldots\ldots,\ b_{60},\ b_{61},\ b_{62}\},\\
D^{64} &= \{b_0, \ldots\ldots\ldots,\ b_{60},\ b_{61}\},\\
&\vdots\\
D^{123} &= \{b_0,\ b_1,\ b_2\},\\
D^{124} &= \{b_0,\ b_1\},\\
D^{125} &= \{b_0\}.
\end{aligned}
$$

These simply correspond to the detector bins which will actually be illuminated by a source at the respective sky element.

The detector array can then be written as,

$$
d_j^k = \begin{cases} b_{N-k+j-1} & ; \quad 0 \le k \le N-1 \quad , \quad 0 \le j \le k \\ b_j & ; \quad N \le k \le 2*N-1 \quad , \quad 0 \le j \le 2*N-k-1 \end{cases}
$$

Physical Mask Aperture Function

Let c_j^k represent the aperture function of the physical mask. And let each of the apertures (mask elements), be represented as, $\{a_0,\ a_1,\ a_2, \ldots,\ a_{61},\ a_{62}\}$. [So a_j, $(\forall\ 0 \leq j \leq N-1)$ will be 1 if it corresponds to an open (transparent) aperture, otherwise it will be 0.]

For different sky elements, following aperture vectors can be listed,

$$\begin{aligned}
C^0 &= \{a_0\}, \\
C^1 &= \{a_0,\ a_1\}, \\
C^2 &= \{a_0,\ a_1,\ a_2\}, \\
&\vdots \\
&\vdots \\
C^{61} &= \{a_0,\ a_1,\ a_2, \ldots\ldots,\ a_{61}\}, \\
C^{62} &= \{a_0,\ a_1,\ a_2, \ldots\ldots\ldots\ldots,\ a_{62}\}, \\
C^{63} &= \{a_0,\ a_1,\ a_2, \ldots\ldots\ldots\ldots,\ a_{62}\}, \\
C^{64} &= \quad\ \{a_1,\ a_2, \ldots\ldots\ldots\ldots,\ a_{62}\}, \\
&\vdots \\
&\vdots \\
C^{123} &= \qquad\qquad \{a_{60},\ a_{61},\ a_{62}\}, \\
C^{124} &= \qquad\qquad\qquad \{a_{61},\ a_{62}\}, \\
C^{125} &= \qquad\qquad\qquad\qquad \{a_{62}\}.
\end{aligned}$$

These correspond to the sections of the mask whose shadow will fall on the detector bin given a source at the respective sky element. The aperture function is then,

$$c_j^k = \begin{cases} a_j & ;\ 0 \leq k \leq N-1 \quad ,\ 0 \leq j \leq k \\ a_{k-N+j} & ;\ N \leq k \leq 2*N-1 \ ,\ 0 \leq j \leq 2*N-k-1 \end{cases}$$

The number of detector bins (and of the corresponding mask elements) illuminated by a source at sky element k is given by:

$$n_k = \begin{cases} k+1 & ;\ 0 \leq k \leq N-1 \\ 2*N-k & ;\ N \leq k \leq 2*N-1 \end{cases}$$

Figure 1.4 shows the mask aperture vector and detector vector for source at sky element k. These vectors for all other sky elements are found out in a similar fashion.

Cross Correlation

With the above definition, we are now ready to discuss the basic steps involved in the cross-correlation procedure. While constructing the sky image, this procedure also accomplishes the subtraction of the mean background level. The following three step procedure undertaken during the cross-correlation achieves this purpose:

1. The *open* or *source* component of cross-correlation, $r_k[o]$, with c_j^k and d_j^k is computed as follows:

$$r_k[o] = \sum_{j \in D^k} c_j^k d_j^k \qquad \forall \quad 0 \leq k \leq 2*N-1, \quad 0 \leq j \leq n_k-1$$

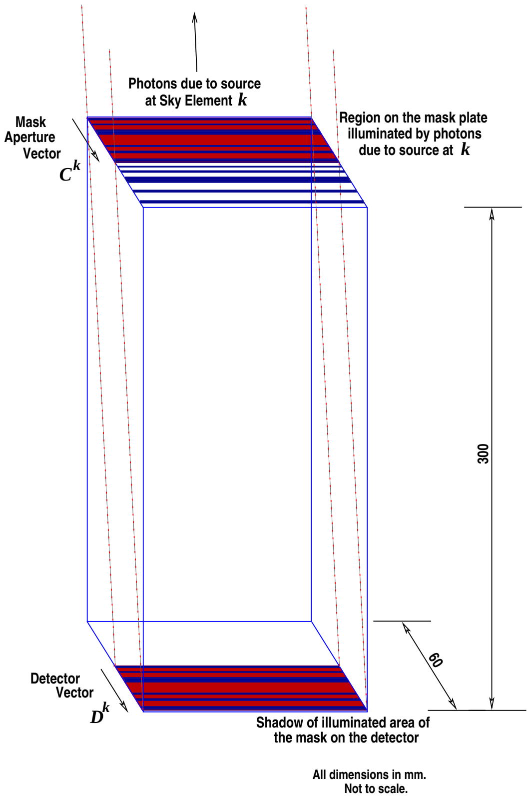


Figure 1.4: *Source at sky element k illuminates a part of the mask plate and a shadow of that part falls on the detector. The mask aperture vector C^k and the detector vector D^k will each have n_k number of elements*

$r_k[o]$ has the information about sources as well as the background.

2. The *closed* or *background* component of cross-correlation, $r_k[c]$, with $\bar{c}_j^k$ and d_j^k is then computed. To construct $\bar{c}_j^k$ [1], the complement of the mask pattern is used; i.e. all open slits are replaced by closed ones and vice versa.

$$r_k[c] = \sum_{j \in D^k} \bar{c}_j^k d_j^k \qquad \forall \quad 0 \le k \le 2*N-1, \quad 0 \le j \le n_k - 1$$

This component responds to the background, and also contains a negative image of the sources.

3. The normalised *closed* component is subtracted from the normalised *open* component, consequently removing the mean background, and adding the source contributions. The result of the subtraction is the image of the sky, r_k.

$$\begin{aligned} r_k &= \frac{r_k[o]}{n_k[o]} - \frac{r_k[c]}{n_k[c]} \\ &= s_k + Coding\ Noise \qquad \forall \quad 0 \le k \le 2*N-1 \end{aligned}$$

where,
$n_k[o]$ $\rightarrow$ the number of open slits in the illuminated area on the mask.
$n_k[c]$ $\rightarrow$ the number of closed slits in that area.
s_k $\rightarrow$ flux due to the source.

[1] $\bar{c}_j^k = 1 - c_j^k$

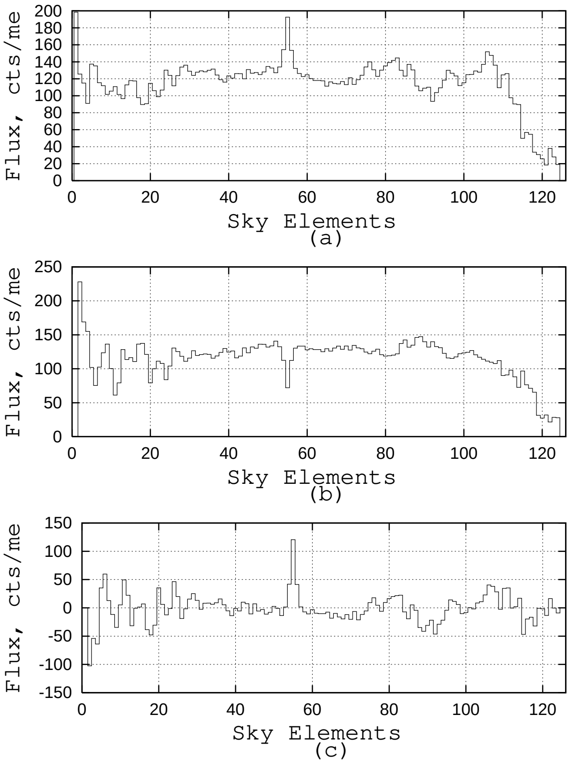


Figure 1.5: *Illustrating the cross correlation procedure. (a) is the source component, (b) is the background component and (c) is (a) - (b), the required image. The source is at sky element 55, and was simulated with finite detector resolution and background noise*

If for instance, there was a source at sky element k, the image shows a sharp peak at k (i.e., s_k) and variable strength peaks at sky elements other than k and this is due to the coding noise. Sources present at sky elements different from k, also illuminate a part of D^k, and result in Coding noise.

Figure 1.5 illustrates the three step procedure of cross correlation.

Reconstruction Array

With a few algebraic manipulations, a new matrix called the Reconstruction array can be built using the mask aperture function, c_j^k, and open fraction t^k, which will simplify the computation of the sky image to a one-step process.

The open fraction of the mask pattern, t^k [that is, the ratio, $\frac{n_k[o]}{n_k}$], is defined as,

$$t^k = \frac{\sum_{j=0}^{n_k-1} c_j^k}{n_k}$$

The sum of the elements of the mask aperture function, c_j^k yields the number of open mask elements in C^k, as open and closed slits are represented by 1 and 0 respectively.

The Reconstruction array, m_j^k is defined as,

$$m_j^k = \frac{c_j^k - t^k}{t^k * (1 - t^k) * n_k} \qquad \forall \quad 0 \le k \le 2{*}N{-}1, \quad 0 \le j \le n_k{-}1$$

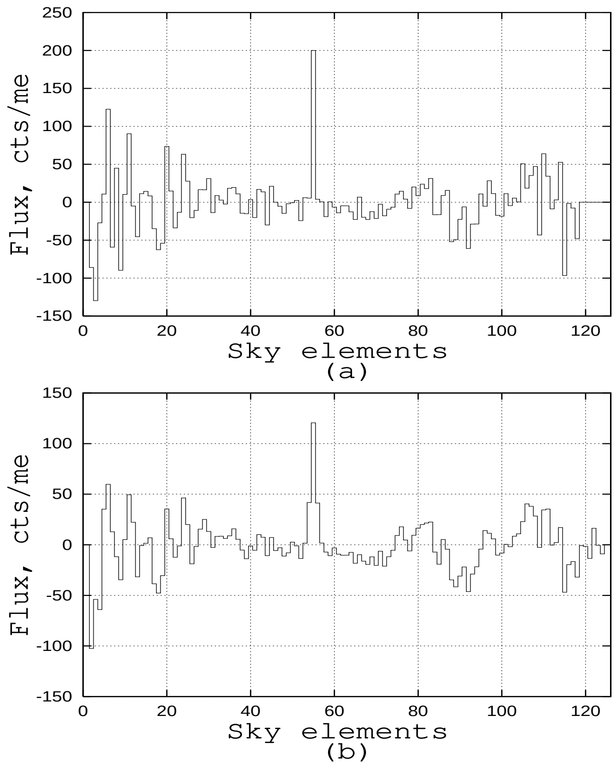


Figure 1.6: *Reconstructed Sky images with source at sky element 55, (a) for an ideal case and (b) for a non-ideal case with finite detector resolution and background noise.*

And the cross correlation between detector array and reconstruction array directly results in the reconstructed sky:

$$r_k = \sum_{j \in D^k} d_j^k m_j^k \qquad \forall \quad 0 \leq k \leq 2*N-1, \quad 0 \leq j \leq n_k - 1$$

where,

r_k $\rightarrow$ Reconstructed Sky obtained after Cross Correlation.
d_j^k $\rightarrow$ Detector Array.
m_j^k $\rightarrow$ Reconstruction(Mask) Array.
D^k $\rightarrow$ Represents the illuminated area on the Detector ($\forall$ j).

Sharp peaks at any sky element in this reconstructed sky image indicates the possibility of the presence of a source. If, for instance, there is a source only at a sky element k, peaks present in all other sky elements other than k, is due to a combination of the coding noise and the Poisson noise. Towards the edges, since the number of mask elements used to normalise the reconstruction array is small, the coding noise becomes significantly high.

Figure 1.6 shows the images obtained using this cross-correlation procedure (between detector array and reconstruction array). The source in image (a) was simulated for an ideal case; image (b) has a source at same location, but simulated with non-idealities of finite detector resolution and background noise. It should be noted that image (b) is same as the image (c) of figure 1.5, which is obtained by subtracting the background component from source component of cross correlation. Also, the plots of figure 1.6 correspond to shadow patterns (a) and (c), in figure 1.3, respectively.

1.1.3 Iterative Removal Of Sources

One is now in a position to find source locations from the reconstructed image. To distinguish source peaks from the coding noise the algorithm of Iterative Removal Of Sources (IROS) is employed. The explanation of the procedure of IROS follows.

The response of the camera for sources at different sky elements is known in the form of Point Spread Function(PSF) for each of them. The procedure to construct the PSFs is as follows. First, a shadow file is created corresponding to the source in a sky element. This shadow file is a normalised description of the shadow pattern devoid of any Poisson noise, background noise or smearing. Since the normalised photon count in every detector bin (corresponding to an open mask element) will be 1, the shadow pattern obtained on the detector will be an exact replica of the mask pattern used. Only in the case of normal incidence, the shadow of the entire mask plate will be falling on the detector; for all other angles, shadow of only a part of the mask plate will fall on the detector. This shadow file is processed through the cross correlation technique, and the resulting image is called the "Point Spread Function" for the sky element considered. The PSF will have a sharp peak of flux 1 ct/me at the sky element which it corresponds to, and will have variable strength peaks at all other sky elements, which represents the coding noise due to this source.

In the image constructed out of simulated or real sky data, the highest peak at any sky element is considered as a candidate source. The strength of this candidate peak can also be estimated from the image. Then to validate that this candidate is indeed a source, we need to check if the RMS value of the image decreases by removing the response due to this candidate peak from the image. For this, the response due to the candidate peak has to be modelled and removed from the sky response. This has to be carried out with the photon-counts data in the bin files. To model the candidate peak, the shadow file corresponding to the sky element at which the candidate peak is present is scaled by the strength of the candidate peak. This response due to the candidate peak is subtracted from the photon-counts in the detector bins, and a reconstruction is made using the resulting new bin-counts. This new image is then examined for a significant decrease in the RMS value.

This removal of the candidate-peak can equivalently be performed in the image domain, instead of the domain of photon counts in detector bins, and this has been adopted in the codes implementing the IROS. In this, to model the response due to the candidate-peak, the PSF corresponding to the sky element at which the candidate-peak was found is scaled by the strength of the candidate-peak and is subtracted from the sky image, thus removing the response due to the candidate peak.

If after removing the response due to the candidate-peak the following RMS improvement criterion is satisfied, that is, if there is a reduction in the RMS value of the image by a percentage of perc (or greater) of that before removal, then the difference between the values of the RMS of the image before and after removal of the response due to the candidate-peak is noted down.

$$RMS_{BR} - RMS_{AR} \leq \mathsf{perc} * RMS_{BR}.$$

where,
$RMS_{BR} \rightarrow$ RMS value of the image Before Removal.
$RMS_{AR} \rightarrow$ RMS value After Removal.
perc is used to specify the percentage improvement in RMS expected.

Then the next higher peak (or less stronger peak compared to the previous candidate-peak) in the sky image is considered for removal. This procedure is carried on until a stage is reached when the removal of the response due to a candidate does not improve the RMS value of the image beyond the level specified. Among the peaks listed till then, the one associated with the largest contributions to the RMS is marked for removal. The list prepared with RMS-differences obtained from the removal of various candidates may also contain entries due to *first-time-negative-peaks*. Such peaks are not considered as valid sources and are ignored. The response due to the marked source is then modelled and subtracted from the sky image. This forms one step in the iteration.

In the next step of iteration, new candidates are looked for from whatever remains out of a *valid* sources' removal in the previous iteration. In every step of iteration one valid source is picked from a list of many probables (candidates). After removing every source, it is checked if some residual (either positive or negative) has cropped up in any of the *already declared* source locations. If any such residual was found, it is modelled and removed without considering the criterion of RMS improvement. The iteration is stopped when, either there is **no** candidate left in the image which can improve the RMS value beyond the specified limit, or if at all there are candidate peaks, all of them are *first-time-negative-peaks*. At the end of iteration a detailed list of all the sources picked up in each step of iteration is displayed; the residuals removed from any location are added to their respective sources and a consolidated and final list of sources is also displayed.

The final image obtained at the end of the iteration, will solely be due to the Poisson noise, which cannot be modelled and removed. One has to note that we only model and remove the coding noise.

Figure 1.7 shows the details of steps involved in Removal of A Source, for an ideal case.

1.2 Two-Dimensional Camera

The procedures of simulation, image reconstruction and iterative removal of sources, explained above are for the one-dimensional case. In our case although we employ 1-d detector (wires) and 1-d masks, there is a possibility of obtaining a limited source location capability across the coding direction. This is due to the following reason. The camera has six mask patterns joined sideways and eight detector wires. The dimensions of the camera are shown in detail in figure 1.8. The wires are placed across the mask slits. The shadow of any of the six mask patterns can fall on the wires. The width of the entire detector (96 mm) is smaller than the width of one mask pattern (125 mm). So in any case, shadow of either one or two adjacent mask patterns will be falling on the detector. Depending on location of the source in the sky, a few wires could be receiving the shadow of one mask pattern and the remaining wires that of the adjacent mask pattern.

1.2.1 Simulation

The photons are generated using the Monte Carlo methods as described earlier for 1-d simulation, the only difference being, instead of a single mask pattern, there will be six here. Among all the photons that pass through the open mask elements and strike the detector, it is necessary to decide which wire do each of these photons get detected by. For this, definite wire-cells are defined around each of the wires, running all along their length. Wires lie at the center of their wire-cells and the width of these wire-cells is same as the wire-to-wire separation(that is 12 mm). A photon is said to be detected by wire W, if the position where it strikes is with in the bounds of wire-cell-W. Again, to account for the uncertainty in reading out the position of photon strike, a Gaussian with a finite σ (of a few mm) is added to the positions of photon-strike before deciding as to which

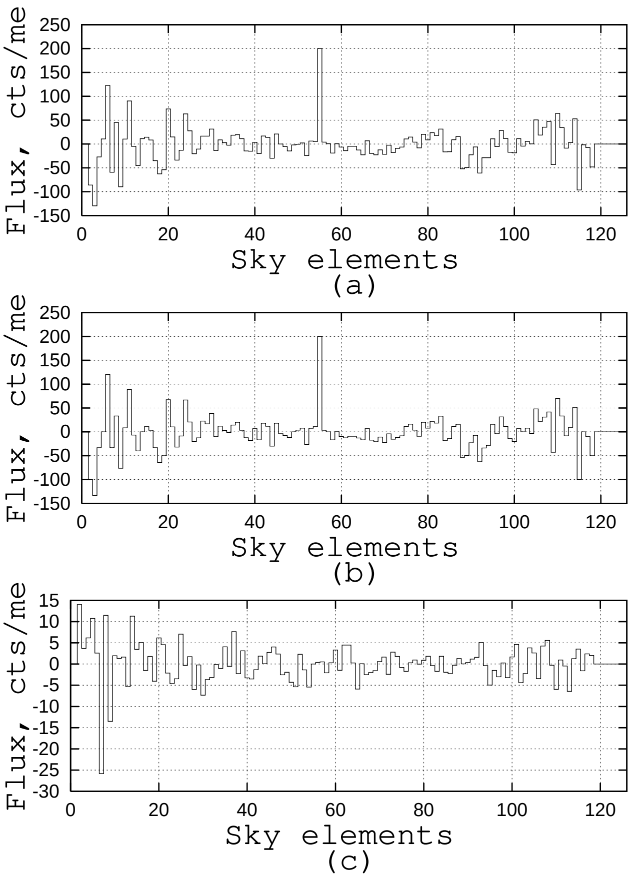


Figure 1.7: *Removal Of A Source. (a) Reconstructed Sky Response for source at sky element 55. (b) Estimated (Modelled) Response, i.e., ($PSF_{@\,k=55}$) * ($Flux_{@\,k=55}$(= 200 cts/me)). (c) Sky Response - Modelled response. Whatever residue remains is due to the Poisson noise.*

wire detects it. A σ of a quarter of the wire-to-wire separation (3 mm) has been used. The σ assumed along the X direction is 0.5 mm, same as in 1-d simulations.

1.2.2 Image Reconstruction

We now extend the concept of sky elements along X-direction also to the orthogonal direction Y. Depending on which mask pattern(s) cast their shadows on which wire(s), the sky elements in Y direction can be defined. For example,
sky element 0 (sky elements along Y are denoted by m) corresponds to mask pattern 1 casting its shadow on wire 8
$m = 1$ corresponds to mask pattern 1 casting shadows on wires 8 and 7......
$m = 7$ corresponds to all wires getting illuminated by shadow of mask pattern 1,
$m = 8$ corresponds to wire 8 getting illuminated by shadow of mask pattern 2, and all other wires by that of mask pattern 1; and so on.
Working out like this, one can list 55 possible sky elements along Y direction.
So $m = 27$ corresponds to normal incidence with shadow of pattern number 3 falling on first four wires and that of pattern number 4 on rest of the wires. $m = 47$ corresponds to all wires getting the shadow of mask pattern 6, and
$m = 54$ (the last one) corresponds to mask pattern 6 casting its shadow on wire 1.

This is also shown in a little more detail below, The first column (m) lists the sky elements along Y direction and the strings (*COMBINATION*) entered across each of the

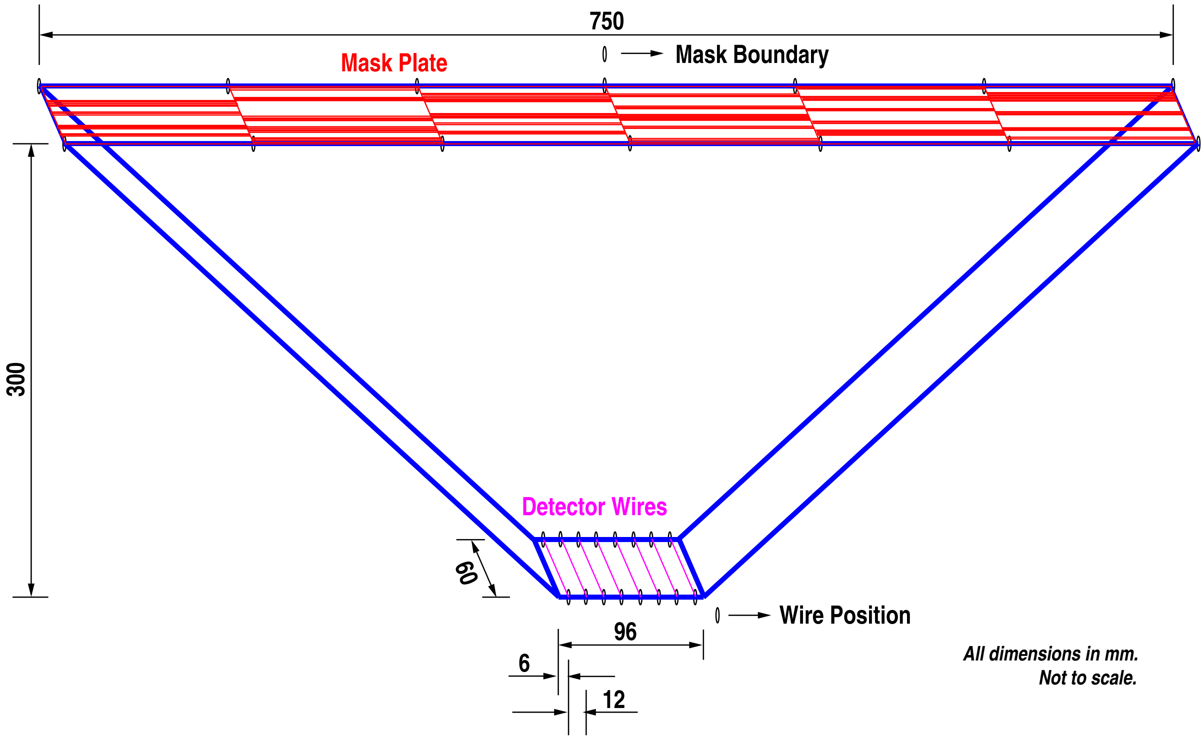


Figure 1.8: *Illustrating the dimensions of the SSM. If viewed in the mask coding direction, it appears to flare out from 96 mm wide detector to 750 mm wide mask plate; and if viewed across the coding direction, it looks like a (300 mm × 60 mm) box.*

sky elements represent the combination of image files that have to be considered. W_8P_1 means image constructed using mask pattern 1 and detector bin counts on wire number 8.

Figure 1.9 shows the method for calculating the angles corresponding to different sky elements along the Y direction.

1.2.3 Iterative Removal Of Sources for Two-Dimensional case

Now, each of the wires will be assumed to have 63 bins. And for each of 8 detector arrays, the cross-correlation is carried out with the reconstruction arrays corresponding to each of the 6 mask patterns [i.e. $8 \times 6 = 48$ independent images]. Then the images corresponding to each of the 55 sky elements along Y-direction are obtained by co-adding images according to the scheme in table 1.1.

The procedure of IROS is similar to that for 1-d case. The candidate-peaks for removal are chosen from these 55 image files. Every time a new candidate is picked, the response due to that candidate is modelled and removed from the sky image response, and the difference in RMS of the sky image before and after the removal of response due to the candidate is noted down. The removal is performed in the detector bin counts for the 2-d case. To model a peak (candidate or a source), the entries in the shadow files for each of the mask-wire combinations corresponding to sky elements at which the peak was found is scaled by the strength of that peak. These are the modelled responses. If for instance, a peak was picked at m, k. The responses due to each of the mask wire combinations (refer table 1.1) corresponding to sky-element-along-Y, m are modelled. That is, the shadow files at sky-element-along-X, k corresponding to the mask patterns involved in each of the

m	$COMBINATION$
0	W_8P_1
1	$W_8P_1 + W_7P_1$
2	$W_8P_1 + W_7P_1 + W_6P_1$
3	$W_8P_1 + W_7P_1 + W_6P_1 + W_5P_1$
4	$W_8P_1 + W_7P_1 + W_6P_1 + W_5P_1 + W_4P_1$
5	$W_8P_1 + W_7P_1 + W_6P_1 + W_5P_1 + W_4P_1 + W_3P_1$
6	$W_8P_1 + W_7P_1 + W_6P_1 + W_5P_1 + W_4P_1 + W_3P_1 + W_2P_1$
7	$W_8P_1 + W_7P_1 + W_6P_1 + W_5P_1 + W_4P_1 + W_3P_1 + W_2P_1 + W_1P_1$
8	$W_8P_2 + W_7P_1 + W_6P_1 + W_5P_1 + W_4P_1 + W_3P_1 + W_2P_1 + W_1P_1$
9	$W_8P_2 + W_7P_2 + W_6P_1 + W_5P_1 + W_4P_1 + W_3P_1 + W_2P_1 + W_1P_1$
10	$W_8P_2 + W_7P_2 + W_6P_2 + W_5P_1 + W_4P_1 + W_3P_1 + W_2P_1 + W_1P_1$
:	:
:	:
:	:
:	:
14	$W_8P_2 + W_7P_2 + W_6P_2 + W_5P_2 + W_4P_2 + W_3P_2 + W_2P_2 + W_1P_1$
15	$W_8P_2 + W_7P_2 + W_6P_2 + W_5P_2 + W_4P_2 + W_3P_2 + W_2P_2 + W_1P_2$
16	$W_8P_3 + W_7P_2 + W_6P_2 + W_5P_2 + W_4P_2 + W_3P_2 + W_2P_2 + W_1P_2$
17	$W_8P_3 + W_7P_3 + W_6P_2 + W_5P_2 + W_4P_2 + W_3P_2 + W_2P_2 + W_1P_2$
:	:
:	:
:	:
:	:
23	$W_8P_3 + W_7P_3 + W_6P_3 + W_5P_3 + W_4P_3 + W_3P_3 + W_2P_3 + W_1P_3$
24	$W_8P_4 + W_7P_3 + W_6P_3 + W_5P_3 + W_4P_3 + W_3P_3 + W_2P_3 + W_1P_3$
:	:
:	:
:	:
:	:
27	$W_8P_4 + W_7P_4 + W_6P_4 + W_5P_4 + W_4P_3 + W_3P_3 + W_2P_3 + W_1P_3$
:	:
:	:
:	:
:	:
31	$W_8P_4 + W_7P_4 + W_6P_4 + W_5P_4 + W_4P_4 + W_3P_4 + W_2P_4 + W_1P_4$
:	:
:	:
:	:
:	:
47	$W_8P_6 + W_7P_6 + W_6P_6 + W_5P_6 + W_4P_6 + W_3P_6 + W_2P_6 + W_1P_6$
48	$W_7P_6 + W_6P_6 + W_5P_6 + W_4P_6 + W_3P_6 + W_2P_6 + W_1P_6$
49	$W_6P_6 + W_5P_6 + W_4P_6 + W_3P_6 + W_2P_6 + W_1P_6$
50	$W_5P_6 + W_4P_6 + W_3P_6 + W_2P_6 + W_1P_6$
51	$W_4P_6 + W_3P_6 + W_2P_6 + W_1P_6$
52	$W_3P_6 + W_2P_6 + W_1P_6$
53	$W_2P_6 + W_1P_6$
54	W_1P_6

Table 1.1: *Images reconstructed using different mask-wire combinations to be considered for obtaining the images corresponding to each of the 55 sky elements along Y direction.*

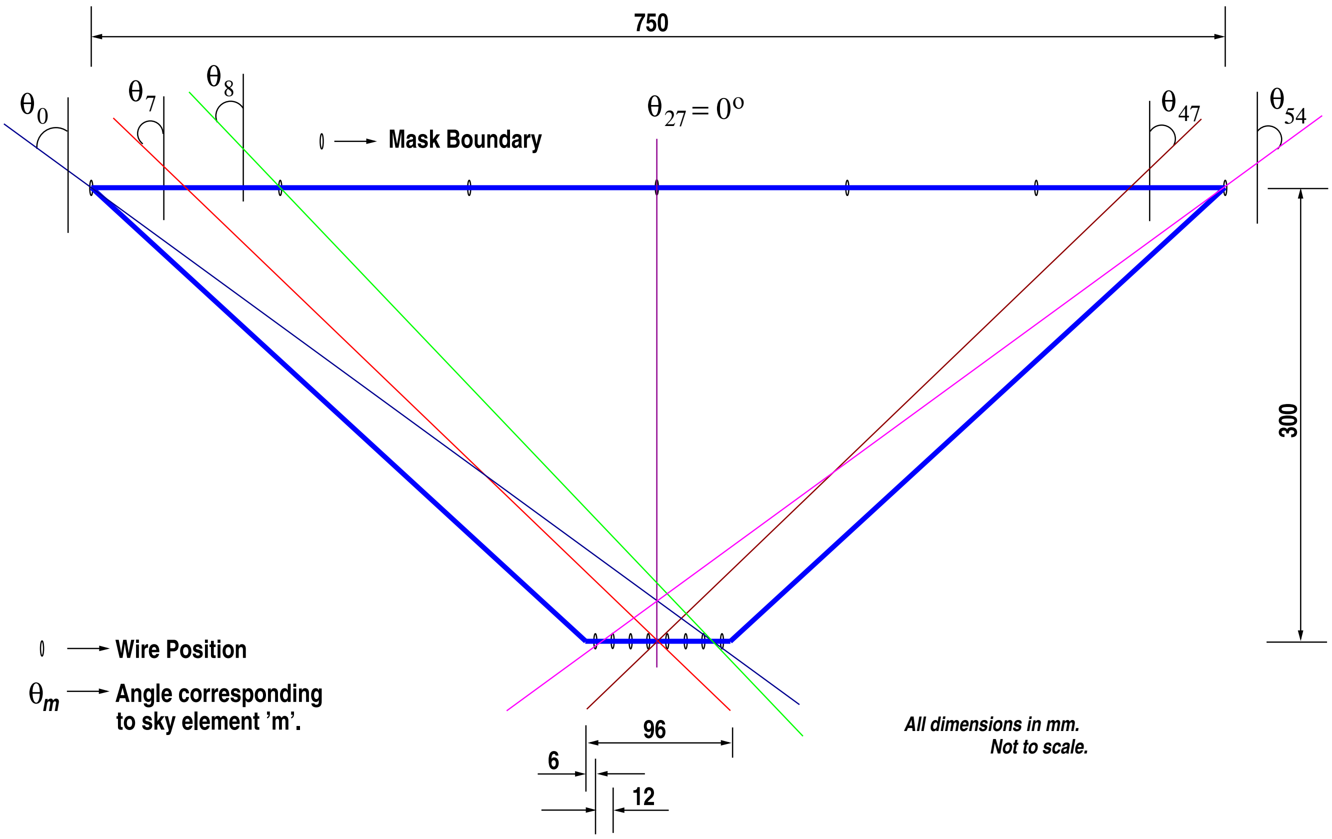
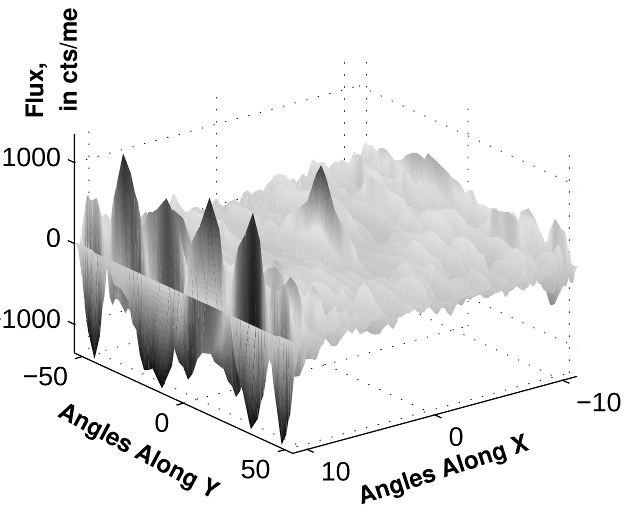


Figure 1.9: *Calculating the angles corresponding to sky elements.*

mask-wire combinations are scaled by the strength of the peak detected. Then each these estimated responses are subtracted from corresponding detector bin counts. Then the images corresponding to all the 55 sky elements are reconstructed. The difference in the RMS values of the image coresponding to sky element m, before removal of the response due to the peak and after it is reconstructed once the response due to the peak is removed, is noted down. As in 1-d case, if the removal of a new candidate picked does not improve the RMS value of the image, no more candidates are looked for in the image. Among all the candidates, using the list of RMS-differences prepared due to each of their removal, the one which gives the best improvement in RMS value is considered as the next source. This selected source is modelled and this response is removed from the sky response, in the manner described above. All the 55 image files are rebuilt once a source is removed. This completes one step in iteration. New candidates are looked for in the new set of reconstructed images.

Every time a source is removed, the residuals that crop up at the previously-declared source locations are modelled and removed, without consideration of the criterion of RMS improvement. Also, the first time negative peaks are avoided even here, as in 1-d case. The procedure of detecting the sources and removing them, go on iteratively, till no more sources can be obtained, i.e. when either no candidate achieves the specified improvement in RMS, or if all the candidates picked are first time negative ones.

In simulations for a 2-d case, to account for the detector and sky background, different noise-values are added to bin-counts on each of 8 different wires. If for a sky image a noise of RMS R has to be added, then, the RMS value of the noise-counts added to bin-values on a wire should be $R/\sqrt{8}$. The image reconstruction and IROS procedures, of course,

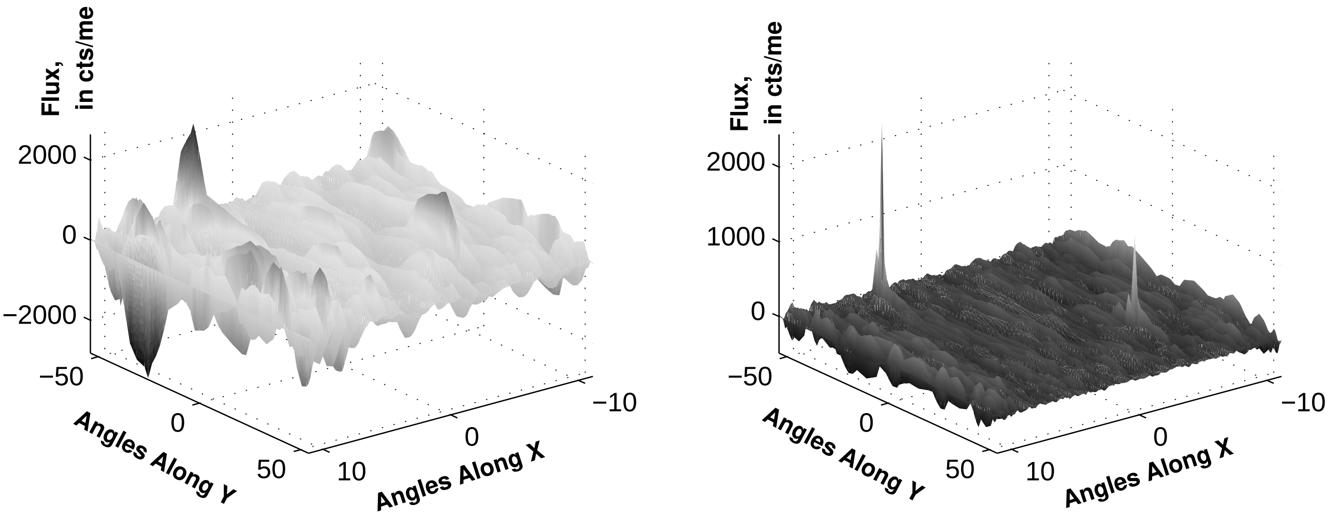


The reconstructed sky image, with a single simulated source, at (X,Y) $= (0.36°, 0°)$.

This source picked up using the IROS procedure. The residual after the entire procedure is shown along with the source.

Figure 1.10: *Single Source*

remain the same. The simulation, image reconstruction, and detection of sources have been carried out for multiple number of sources too. Figures 1.10 and 1.11 show the plots for 2-d case, for a single source and for two sources, respectively. Both the examples are for a detector with finite resolution, and a background noise of RMS value 100 cts/me. In the second example with two sources, the source at (X,Y) $= (4.9°, -42.4°)$ (i.e., $k = 35$ & $m = 45$), has been simulated to have twice the strength as that at (X,Y) $= (-4.9°, 20.6°)$ (i.e., $k = 90$ & $m = 20$).



The reconstructed sky image, with two simulated sources at $(X_1, Y_1) = (4.9°, -42.4°)$ *and* $(X_2, Y_2) = (-4.9°, 20.6°)$.

These two sources recognised, using the IROS procedure, shown here with the residual.

Figure 1.11: *Two Sources*

Chapter 2

THE SOFTWARE

All the codes have been written in C. To get a small description about any code, one can type "desc <name of the code>" and if the code receives any command line arguments, just typing the name of the executable will give a list of arguments that it expects. Here are brief descriptions of each of the codes.

2.1 /usr/home/astrosat/RAVI/

All the codes have been written in C. The directory structure of all these codes is as follows. The directory, */usr/home/astrosat/RAVI/* contains the simulation codes, *Chk.positions.c*, *Chk_ran_positions.c* and *positions_actual.c*, for a 1-d case. As mentioned earlier, Monte Carlo techniques have been employed for writing simulation codes. All of these codes receive the source angle (along X direction) and σ_x [to be added to the detected photon positions] as input arguments.

2.1.1 positions_actual.c

positions_actual.c reads the mask pattern from a file, *mask_pattern_actual.data*. At present this is a copy of *mask_pattern1.data*: the mask patterns are stored in files called *mask_pattern#.data*, where # can be any digit between 1 and 6. This program stores the slit pattern (represented by 1 for an open slit and 0 for a closed one), in an array. From the positions where photons strike on the mask plate (these positions are found out using the procedures, *gauss_dev_func.c* and *ran_gen_func.c*), the index to this array is calculated and the photon is allowed to strike the detector (at the source angle), only if that position where photon strikes the mask plate corresponds to an open slit. Specified number of photons are generated, and the positions where the photons (which would have passed thro' an open mask slit) strike the detector are stored in an output file in the same directory (*RAVI*). The names of these output files are generated using the value of the σ and the source angle given as input data.

2.1.2 Chk.positions.c

Chk.positions.c differs from *positions_actual.c* in two things. In *Chk.positions.c* the mask pattern to be considered for simulations can be specified (as another input argument). And the other difference is that it expects mask-slit-boundary-file (generated using *Chk.put_bound.c*) instead of the slits themselves. The code, *Chk.put_bound.c* simply reads the specified mask pattern file, *mask_pattern#.data*, and generates a boundary file for that mask pattern. This boundary file will have the positions of the boundaries of the mask slits when there is a transition from an opened slit to a closed one, or vice versa. These positions will be present in the first column of the boundary file, and across each of these positions of boundaries, in the second column, 1 will be entered if there is a transition from a closed slit to an opened one, or 0 will be entered when there is a transition from opened slit to a closed one. Clearly, there will be alternating entries of these flags 1 and 0 in the second column.

2.1.3 Chk_ran_positions.c

The other simulation code, *Chk_ran_positions.c* is a modification of *Chk.positions.c*, written to find out the maximum tolerence limit in the case of the non-ideality of unevenness in the widths of the mask slits. It also receives the mask-boundary files as the input, and adds prespecified percentage of error (which is another input argument to this code), to the $mask-slit-boundaries$.

2.1.4 gauss_dev_func.c, ran_gen_func.c

The procedures, *gauss_dev_func.c* and *ran_gen_func.c* have been implemented directly from the Numerical Recipes. *ran_gen_func.c* is a random number generator (random number lies between 0 and 1; these limits not included), using the Park and Miller minimal standard algorithm, with shuffling. And the procedure *gauss_dev_func.c* generates a Gaussian distributed deviate. It internally uses the random number generator in *ran_gen_func.c*.

2.1.5 histogram.c, get_angles.c, desc.c

The code *histogram.c* is used to make histograms of some data. It expects the configuration options, such as, number of histogram bins, position of the first bin, that of the last bin, and a flag (1 or 0), to decide on out-of-bound values to be present (in the order written above) in a file called, *histogram.cfg*. The data which have to be binned will be read from the standard input, and the output will be written on the standard output. The code, *get_angles.c* produces the values of the source angles corresponding to sky elements, 0 to 62. The output is stored in a file called, *angles.data*. The code *desc.c*, reads the given C code file, and searches for the string, "*desc*>". Then goes on printing the text following *desc*>, either till it reaches another string, "<*desc*" or *eof*. Every C-code written has a few lines of brief description at the top, telling what exactly that particular code does and how to use the executable. The appendix of this report contains archive of such descriptions of all the relevant codes.

2.2 /usr/home/astrosat/RAVI/RECONSTRUCTION/

2.2.1 detector_bins.c, Chk.detector_bins.c

The codes in the directory, *RECONSTRUCTION* handle the image reconstruction, for a 1D case. The code, *detector_bins.c*, creates the detector bins (number of bins is 63, selected to be equal to the number of the mask elements), using the simulated detected positions (along X-direction), present in files whose names start with "*det_pos_*" and stored in the directory *RAVI*. The output is stored in the same directory as the executable, that is *RECONSTRUCTION*, in files whose names start with "*bin_*". And rest of the name is constructed again using the values of σ and source angle (as was the case when generating the name of the simulated-data-file). The code, *Chk.detector_bins.c* can also be used for generating the detector bin files using the simulated data. It differs from *detector_bins.c* only in the aspect that the user is expected to specify the names of the input and output file names, instead of generating them using the values of σ and the source angle (as what the code *detector_bins.c* does).

2.2.2 recons_array.c

The code, *recons_array.c* generates the reconstruction arrays for the mask pattern used. The mask pattern file, *mask_pattern_actual.data* which this code uses is a copy of the file, *mask_pattern1.data*. Each of the output arrays (corresponding to different sky elements along X direction) are stored in files, "mask#", where # is an integer representing the sky element which the file corresponds to.

2.2.3 detector_array.c, reconstn.c

The code, *detector_array.c*, produces the detector arrays using the bin file produced. 126 different output files (names of the output files start with "det", and it is followed by an integer representing the sky element the file corresponds to) are produced, and stored in the same directory as the executable. The code can also receive a shadow file as the input, instead of a bin file. The code, *reconstn.c*, is the one in which the cross-correlation algorithm is implemented. It assumes the detector arrays and the mask arrays to be present in files named det** and mask** (** represents the sky element the file corresponds to) respectively. The execution of this code should follow the execution of *detector_array.c*; those det-files may get over-written with data for some other source. The code writes the output to the standard output. The output image is usually stored (by redirecting the output from standard output) in a file whose name starts with "*p_*". And the names of these files have been derived from the bin files they correspond to.

So for instance, if the simulated X-positions of the photon-strike were stored in a file called, "*det_pos*_0.5_4.00", it means that the value of σ used during simulations was 0.5 mm, and the source angle is 4.00° (this is rounded-off to two decimal places, so if the simulated source angle was 3.9956°, or 4.0023°, the output file name will be "*det_pos*_...._4.00" only). Then the bin-file will be "*bin*_0.5_4.00", and the corresponding image file will be "*p*_0.5_4.00". [Both of these bin file name and the image file name can be user-specified, but this is the convention followed.]

2.2.4 noise_files.c, noise_bins.c, add_noise.c

The code, *noise_files.c* generates different noise files (in revision, 1.3, six noise files are generated). The noise values are Gaussian deviates, generated using the procedure, *gauss_dev_func.c*. The RMS-value of the noise files so generated will be unity. The code *noise_bins.c* is used to scale the RMS value of the noise files. Whenever noise has to be added in the simulations, the RMS-value of the noise files will have to be scaled and then added to the source-bin-counts. This job is done by the code, *add_noise.c*.

2.2.5 scale_bins.c, add_n_bins.c

When multiple sources have to be simulated the bin-count-files produced for individual sources will have to be added. The code *scale_bins.c* can be used to scale bin-counts due to a source and add the scaled bin-counts to bin-counts due to another source. The code, *add_n_bins.c* is capable of doing this for N number of sources. It expects the name of a 'map-file' which should contain the names of the bin-count-files in first column and the scaling factor (by which the bin counts will have to be scaled) in the second column across each of the bin-file-names. In that way any number of sources can be simulated in one sky image.

2.2.6 pt_sprd_funcs.c

The code, *pt_sprd_funcs.c* produces the shadow files (which are used to generate the point spread functions at each of the sky elements along X-direction). The shadow files are simply the shadow of the mask pattern on the detector, shifting along the detector at the angle corresponding to the sky element. [For producing the PSF file from the shadow file, the name of the shadow file will have to be given as input to the code *detector_array.c*, and the code, *reconstn.c* should be invoked, to get the corresponding PSF.]

2.2.7 rmpsf.c, rmsrc.c

The idea of removal of sources has been implemented using the codes, *rmpsf.c* and *rmsrc.c*; but they dont go iteratively checking the residuals, or multiple sources. The user has to pick up peaks from the image and specify those to these codes. Main differences between these codes are as follows. *rmpsf.c* expects the location of the peaks as well as the flux that has to be removed, to be specified. And these things have to be specified in a map file, which the code will read. The end product after having removed all the sources specified is written on the standard output. The code, *rmsrc.c* expects only the locations of the peaks (that should be removed) specified, and that too on the command line. Each of the intermediate output obtained after removing every source (at locations specified on the command line) is written in separate files, whose names are generated using the fluxes removed and the locations at which they were present.

2.2.8 mean_rms.c

The code *mean_rms.c* reads the data present in the standard input and produces the mean and the RMS values of that set. If the rms and/or mean of the fluxes from image files have to be found out, the code can be asked to neglect the fluxes due to strong source-peaks by specifying their locations on the command line.

2.3 /usr/home/astrosat/RAVI/TWO_DEE/

2.3.1 det_positions.c

The directory, *TWO_DEE* contains mainly codes related to **2-D** case. But it also has a few codes written for a **1-D** case. The code, *det_positions.c* simulates a 2-D camera. Now there will be six mask patterns stacked one next to the other. And in the software these are stored in a two-dimensional array, whose first dimension takes care of the pattern number and the second dimension specifies the serial number of the slit in the mask pattern. As in the 1-D case, specified number of photons are generated, and made to 'fall' on this mask plate. Two indices along Y (giving the pattern number) and along X (which gives the slit number) are found out from the position where the photon strikes the mask plate. If this position corresponds to an open slit (of any of the six patterns), it is allowed to strike the detector at the source angle (along X and along Y). Again a definite σ is added to the detected position along both X and Y directions. The σ along Y direction is selected to be a quarter of the wire-to-wire separation, which is 3 mm. The σ along X has been taken to be 0.5 mm. The X-positions at which the photons strike and the wire number that position corresponds to, are written in two columns in the output file, which is put in a sub-directory, "*DET_POS*". And again the name of the output file is constructed using the values of σ's and angles (along X and Y), *det_pos_*$<\sigma_x>$_$<\sigma_y>$_$<\mathit{angleX}>$_$<\mathit{angleY}>$ [one decimal place of the σ's are used, while two decimal places of the angles are used to frame the output file name]. And in order to get the random numbers and the gaussian deviates, the procedures in the files, *gauss_func.c* and *ran_num_func.c* are used by *det_positions.c*.

2.3.2 wire_positions.c

The output file of the simulation code, *det_positions.c* will have the X-positions of the photon strike in the first column, and the wire number corresponding to that X-position in the second column. The code, *wire_positions.c* receives this file (just specifying the name of the file is enough, the code searches for that file in the directory, *DET_POS*), as input and lists out the X-positions on each of the wires in eight different files. Means picks up all the X-positions in the file given, which are falling on wire number # (# can be 1,2,3,...8), and stores those positions in a new file created for wire number #. These output files are created by appending "_wire#" to the name of the input file given to it.

2.3.3 create_bins.c

The rest of the procedure till image reconstruction is similar to that for a 1-D case. The X-positions on each of the eight wires are binned and so there will be eight bin-files corresponding to each of the eight wires. This task is accomplished by the code *create_bins.c*. It can process one X-positions file at a time, so has to be invoked eight times to get the bin files on all the wires. It expects both the input and output file names to be specified, searches for the input file in the directory, *DET_POS* and stores the output file in the directory, *DET_BINS*. If the input file has X-positions on any wire (that is if it was generated by *wire_positions.c*), then it will have a single column with only the X-positions, but if it is the output of say *positions_actual.c*, there will be two columns. So if the input file has a single column of X-positions, an additional argument, "-w" will have to be given on the command line. [This could be avoided by using "fgets" on the input file, instead of "fscanf"].

2.3.4 create_det_array.c

The detector arrays are generated by the code, *create_det_array.c*, which is very similar to *detector_array.c* present in *RECONSTRUCTION*, except for the few things that this code searches for the bin files in the directory *DET_BINS*, and for shadow files in the directory *SHADOW#* (# specifies the mask pattern number, and so it can be 1,2,.... or 6). The user has to specify whether a bin file has to be processed or a shadow file, by using the switches, "-b" and "-s" respectively. If shadow file has to be used, the mask pattern number (to which the shadow files belong) also needs to be specified. The detector arrays are stored in the directory, *DET_ARRAYS*, in files, det##, where ## indicate the sky element number, the array corresponds to.

2.3.5 x_correlate.c

The execution of the code, *x_correlate.c* should follow the execution of *create_det_array.c*. It accesses the detector arrays in the directory, *DET_ARRAYS*, and cross correlates it with each of the six reconstruction arrays generated for each of the six mask patterns. These reconstruction arrays are present in the directories, *RECONS_ARRAY#*, # specifying the mask pattern number, 1,2,...6. [The reconstruction arrays are generated using the code, *gen_recons_arrays.c*]. So, for a set of detector arrays, generated for a given bin-file on a wire, six images will be produced. The name of the sub-directory under which all these image-files have to be stored, will have to be specified as the (only) argument to this code. It creates this sub-directory (if it is not existing already), under the directory, *PLOTS*, and stores the six output files in it. These six output files are named *pattern*1, *pattern*2,....*pattern*6.

2.3.6 images_y.c

For eight wires, totally 48 images will be generated. [If the source was present towards the edges, along the Y-direction, a few wires may not get any photons at all, even in such cases, images are produced, but will have zeroes entered at all the sky elements]. And there are 55 combinations of these images possible, corresponding to 55 different sky elements along Y direction as has been mentioned already. These 55 image files are produced by the code, *images_y.c*. This code assumes that the sub-directory for some wire# under *PLOTS* (having files, *pattern*1, *pattern*2,.... *pattern*6) will have names, $@@&&$_*wire#*. The argument to this code should contain the initial part of the name of the sub-directory, i.e., $@@&&$. The code itself appends "_*wire#*", (# = 1,2,...or 8), to the string, $@@&&$, and accesses all the 48 files (named *pattern*1, *pattern*2,....*pattern*6, in eight sub-directories). A separate sub-directory, by name, $@@&&$ is created under *PLOTS* (if not created already), and all the 55 output image files are stored under it. These files are simply named 0,1,2.....,54, after the sky elements (Y), they correspond to.

2.3.7 locate_srcs_1By1.c

The next step is that of the Iterative Removal Of Sources, IROS. The code, *locate_srcs_1By1.c* implements that algorithm. All the 55 image files constructed by *images_y.c* (and stored under *PLOTS*) are copied to a sub-directory (having same name as that in *PLOTS*) under the directory *TRASH*. And the bin-files are copied from *DET_BINS* to the files with same name to the directory, "*TRASH/BINS/*". All the 48 image files corresponding to 8 wires and 6 mask pattern, are stored in the array, *ImagesOnWires*. And all the 55 image files corresponding to 55 sky elements are stored in the array, *Images*. [This is also copied to another array, *Images_org*, to be used later to prepare the matrix, *ORG_MATRIX*.] The detector bin counts are stored in the array, *Bins*. The array, *MaskArrays* will have the Reconstruction array entries for all the mask patterns, loaded from the files, *RECONS_ARRAY#/mask%* [# can be 1,2,... or 6, representing the mask pattern number and % can be 0,1,2,... or 125, representing the sky element(X) number]. The array, *DetectorArrays* will hold the values of the detector arrays, for all the wires, while reconstructing images after a source-removal. The array, *Shadows* will have the shadow files for all the mask pattern, and are loaded from the files, *SHADOW#/sha%* [# : 1,2,... or 6, (mask pattern number) and % : 0,1,2,...or 125, (sky element (X) number)]. The array, *Combn*, will have various strings loaded from the file, *sky_elem_y_ip.data*, for each of the sky elements along Y direction; these strings give an idea about how many and which all files (out of 48 images) have been added to get those 55 images. All these multidimensional arrays are declared to be global arrays, and are accessible by all the sub-routines used in this code.

find_peaks_func_1By1.c

Out of all the available 55 image files, the one having the highest peak is found out, using the subroutine, *find_peaks_1By1()* (present in the file, *find_peaks_func_1By1.c*). It searches thro' the array, *Images*, and lists out the highest peak in each of the 55 image files, and picks up the highest among these 55 peaks, and puts the flux and the location (along X and Y directions) in the handles provided by the calling procedure. This peak is only a candidate for the next-source. The RMS improvement resulting by removing this peak is noted down by the *main()* procedure. All these information about the first peak returned by $find\_peaks\_1By1()$ is noted down in a structure, *List*, anonymously accessed by a pointer, *lp*. This structure has four members, Sky element X, Sky element Y, Flux and the Difference in RMS (before and after removing the source). The RMS value is calculated using the procedure, $rms()$ [present in the same file, *locate_srcs_1By1.c*]. All the first-time-negative peaks are not even considered for checking the RMS improvement. The procedure, $trap\_a\_source()$ [also present in *locate_srcs_1By1.c*], is used to model this candidate-peak, and remove it. By modelling what is meant is, the shadow file corresponding to the location of the peak given to it, is scaled by the scaling factor given, and this is called, modelled response. This scaling factor is calculated by dividing the strength of the peak (to be removed), by the number of image files (out of 48 files) added to get the image file (one among 55, in which the peak was found); this number can be determined from the strings in the file, *sky_elem_y_ip.data*. But it is also known that for sky elements(Y), lesser than 7 the number of files involved is one more than the sky element number, for sky elements greater than 47, it is 55 minus the sky element number, and for the rest of the sky elements, there are 8 files involved. This apriori knowledge is used in the code to get this multiplication factor. $trap\_a\_source()$ always removes the modelled response due to any peak given to it, from the array, *Bins*, and stores that resultant in another array, *Bins_spare* [which has the same dimensions as *Bins*]. This modelling-removal is carried on for all the files contributing in the image file where the peak was reported to be present.

For example, say if the peak was reported at, $SkyElemY, m = 27$, and $SkyElemX, k = 60$. The image file at $m = 27$ is produced by adding the eight image files, one due to each of eight wires. And images on wire numbers 1 to 4 will be produced by shadow of pattern number 3, and those on wires 5 to 8 will be produced by shadow of pattern number 4. And there will be a peak in each of these eight constituent files at $k = 60$. So for removing the peak at $m = 27, k = 60$, the peak at $k = 60$ in each of the eight constituent files are modelled, by scaling the corresponding shadow files by the given scaling factor, and this modelled response is subtracted from the sky response, in array, *Bins*, in relevant wire-pattern combinations. The resultant of this subtraction is stored in *Bins_spare*, again in corresponding wire-pattern combinations. The resultant is stored in *Bins_spare* because only a candidate peak is removed, and it is not yet validated as a source.

If in case the source was present towards the edges along Y [that is if sky element (Y) was lesser than 7 or greater than 47], a few of the wires may not be getting illuminated at all, and this is also evident from the string combinations, in the file *sky_elem_y_ip.data*. In such cases, for those non-illuminated wires the entries in *Bins* will be copied to *Bins_spare* unaltered.

create_det_arrays_irs_1By1.c, x_correlate_irs_1By1.c, images_y_1By1_func.c

Once the new bin-files are obtained, (in order to find out RMS value in the image file after having removed the candidate peak) the image files are reconstructed, using the subroutines, $create\_det\_arrays()$ and $x\_correlate()$ [present in files, *create_det_arrays_irs_1By1.c* and *x_correlate_irs_1By1.c* respectively]. $create\_det\_arrays()$ generates the detector arrays for all the wires, using each of the eight bin files present in the array, *Bins_spare* [in which the resultant of shadow-file removal is stored]. The detector arrays are stored in the array, *DetectorArrays*, for all the wires. The procedure, $x\_correlate()$ uses the Reconstruction arrays in the array, *MaskArrays*, and the detector arrays in the array, *DetectorArrays*, and generates all the 48 images, corresponding to eight wires and six mask patterns. These images are stored in the array, *ImagesOnWires*.

Then, the RMS value of the image in which the candidate peak was reported is found out again, using the procedure, $GetImageRMS()$ [present in the file *locate_srcs_1By1.c*]. $GetImageRMS()$ *co-adds* only that image in which the source was present, and calls *rms*() to get the RMS value of that new image (obtained after removal of the candidate-peak). If there is an improvement in the RMS value of the image, by removing the candidate peak, [which is decided on the basis of percentage improvement of RMS; if the percentage of the difference between these RMS values, (before and after subtraction of the modelled response), is greater than *perc*% (*perc* is an input argument to the code), it implies that, that candidate peak is a valid one], then this difference in RMS values is noted down along with information about the location and strength of the candidate peak in the structure *List*, (accessed using the pointer, *lp*), and $find\_peaks\_1By1()$ is invoked again to find the next candidate-peak. This time, $find\_peaks\_1By1()$ checks out the entries made in the structure, *List*, and avoids reporting them [since they have been already considered]. It is to be noted that this structure, *List* contains information about all the candidates, first-time-negatives, positive ones improving RMS (which are the ones likely to be considered as the next valid source), valid residues (positive and negative) of the previously removed sources.

This listing of all the candidates go on till a point is reached when a candidate peak does not improve the RMS value. This candidate peak will NOT be a first-time-negative, and could be a residue of any of the previously-reported sources, or may be a peak at a new location, where no source has been reported till then. This peak is rejected (that is, not noted down in the structure, *List*), assuming that (the noise-level has been reached, and that) any other peak that will be picked, beyond that point, is also not going to improve the RMS value. Once such a RMS-not-improving peak is encountered, the span of sky elements(X) in which the candidates were looked for is checked. If it has to be expanded, it is done, and again, candidates are picked from this expanded span of sky elements (X). [span of the sky elements (X), is expanded like this in four steps, (31-93), (16-108), (16-116), and finally (0-125)]. When even this final span is reached [when a RMS-not-improving peak is encountered], the *main* procedure checks out all the entries in the structure, *List*, and picks up the peak, removing which maximum difference in RMS is observed. This peak will be the valid source in this step of iteration. This is again modelled and response removed from the sky response. This time, after calling *trap_a_source*(), $images\_y\_1By1()$ is called. The procedure, $images\_y\_1By1()$ [present in the file, $images\_y\_1By1\_func.c$, uses the 48 images present in array, $ImagesOnWires$ and produces the 55 images corresponding to each of the 55 sky elements along Y direction. These files are produced with the simple observation (refer file, *sky_elem_y_ip.data*) that, to produce images for any of Sky Element Y (m) = 1 to 7, one image file (out of 48) has to be added to the image corresponding to the previous m. For getting images for any of $m = 8$ to 47, one image file (out of 48) has to be removed from the image corresponding to previous m, and one another (out of 48), has to be added to it. And for getting images for any of $m = 48$ to 54, one image (out of 48) has to be subtracted from the image corresponding to previous m. Then another procedure, *reload*(), (also present in *locate_srcs_1By1.c*), is called, and it copies all entries in the array, $Images\_spare$ to $Images$, and that in $Bins\_spare$ to $Bins$. This is to ensure that new sources are looked for (and removed from), whatever has remained out of previous sources' removal. The subroutine also writes this resultant of a valid source removal, in files, as the next version. Under the sub-directory (under *TRASH/*), having all the 55 image files with raw data, a sub-directory, by name, *perc*#.####, where #.#### is the value of the *perc* used in the code, is created. Under this sub-directory, another level of sub-directories are created representing the version numbers. [they are named $v1$, $v2$, etc]. In these sub-directories, [$v1$, $v2$, etc], the image files obtained after removing every source (both 48 files corresponding to 8 wires and 6 mask patterns; and also 55 images corresponding to 55 sky elements] are stored in their respective version-directories. In the *main* procedure, the information about the valid source removed is also stored in another structure, called *Log*, anonymously accessed by the pointer, *gp*. The location of the source (along X and Y), the source strength, and the RMS values before and after removal of the peak are all stored in this structure. [to be used to check if any peak reported later is a residual of the source reported already, and also to list out all the sources picked at the end of the entire IROS procedure].

The source removed presently may have contributed to the strengths of the sources removed in the previous iterations. So when it is removed it may result in a (positive or negative) residual flux at (any or all of the) locations of previously detected sources. So, after removing every valid source (one selected from the list of all the candidate-peaks), the fluxes at locations of all the previously reported sources are checked, to see if they have crossed a threshold [a macro, THRESHOLD, #define'd to 10, in revision, 1.4.1.3 (of *locate_srcs_1By1.c*, is used for this purpose]. If the residual is substantial, then it is modelled and removed, without checking for RMS improvement (since it is already known that a valid source exists at that location).

Then, a fresh start is made to check out all the candidate peaks from whatever remains out of a source's [or the residuals' (at locations of previously declared sources)] removal. This procedure goes on until, there is not even a single valid (RMS-improving) peak (candidates) picked, or if all the peaks picked are first-time-negatives. Finally, all the sources picked are listed on the standard output, and also written in some files (named as *DETAILS*, which lists information about each step of iteration and *SOURCES*, which gives a consolidated picture about the sources), under a separate sub-directory (called *perc#.####_RESULTS*, created in the directory containing 55 image files with raw data, and hence present in the same directory where *perc#.####* is created). Also in the same sub-directory, few matrices are stored. One with the raw data, before IROS is employed (in file named, *ORG_MATRIX*), another with only the fluxes (of sources picked up), at their respective locations and zeroes in all other (non-source) locations (stored in file called, *MATRIX*), one another matrix, with the source strengths picked, added to the residual (that remains after the entire IROS is completed, at all the locations) is stored in *RES_MATRIX*. It is assumed that the way in which the source-flux spreads out across adjacent sky elements follows a Gaussian distribution, and a convolution algorithm is employed on these matrices to take this feature in to account. The convolution is employed on *MATRIX* and *RES_MATRIX*, and the respective output matrices are stored in files called *MATRIX_CONV* and *RES_MATRIX_CONV*. The procedure, *convolution*() [present in the same file, *locate_srcs_1By1.c*] is used for this purpose.

2.3.8 gen_noise.c, add_noise.c

There are eight noise files, generated using the code, *gen_noise.c*. These noise files are simply gaussian deviates (obtained using the procedure in the file, *gauss_func.c*), generated using different seeds. The noise files are named, *noise*1, *noise*2,..... *noise*8, and are stored in the sub-directory, *NOISE*. When noise has to be added to the simulated data, the code, *add_noise.c* has to be used. This code expects the name of the bin file (to which noise has to be added) be specified, along with the scaling factor by which the RMS value of the noise files (which will originally be unity), have to be scaled. The name of the bin file should not contain the trailing "*_wire#*" part(where # can be 1,2,.... or 8, representing the wire number). The code itself appends this part, *_wire#*, to the given bin-file name, and adds noise to bin files on each of the eight wires. The code uses the file, *noise*1 for bin file on wire number 1, *noise*2 for bin file on wire number 2, and so on, adding different noise-files to each of the wires. The names of the output bin files (with noise added) are obtained by inserting "*_S@.@@*" (where @.@@ is the scaling factor given to the code) before the wire part, (i.e. *_wire#*, with # being 1,2,... or 8) in the names of corresponding bin files. The input bin files are looked for in the directory, *DET_BINS*, and output files are also stored in the same directory. The same IROS code, *locate_srcs_1By1.c*, can be employed on this non-ideal case of bin files with noise.

2.3.9 locate_srcs.c, locate_srcs_NoiseFactor.c, locate_srcs_RMS. c, create_det_array_irs.c, x_correlate_irs.c

Apart from *locate_srcs_1By1.c*, which is the latest version among codes handling the IROS (as on April '02), there are many older versions, like, *locate_srcs.c*, *locate_srcs_NoiseFactor.c* and *locate_srcs_RMS.c*. And all of these mainly use file I/O (as opposed to memory I/O in *locate_srcs_1By1.c*). *locate_srcs.c* simply assumes the highest peak to be the next source. The iteration goes on until the highest peak picked up (by the procedure, *find_peaks*() in the file, *find_peaks_func.c*) is lesser than a threshold (hard coded as 10). And if noise had been added during the simulations, the iteration is stopped if the peak detected is lesser than twice the RMS value of the noise added. The code *locate_srcs_NoiseFactor.c* varies from *locate_srcs.c* mainly in the case when noise had been added during the simulations is handled. *locate_srcs_NoiseFactor.c* compares the peak picked with NoiseFactor times the Noise-RMS, instead of twice the Noise-RMS (as in *locate_srcs.c*). The code *locate_srcs_RMS.c* uses one another criterion for deciding the validity of the peak picked and to decide the point when the iteration has to be stopped. It compares the RMS value of the image file (in which the peak was found), before and after the removal of the peak. If there is a *significant* improvement in the RMS value, then that peak is validated to be a source, otherwise other peaks of lesser strengths are taken up for checking the the RMS improvement due to their removal. And the iteration is stopped when there is no peak left which improves the RMS (by its removal). And this is valid for both the cases, simulations with noise and without noise. And in all the three cases, while producing the bin files and image files when the iteration is being carried on, to reconstruct the images after removal of every source, the procedures in the files, *create_det_array_irs.c* and *x_correlate_irs.c* (which are modified forms of *create_det_array.c* and *x_correlate.c* respectively) are used.

2.3.10 Chk.1By1.1d.det_positions.c

The directory, *TWO_DEE*, also has complete set of codes handling the 1-D case, right from simulations to those implementing the IROS. *Chk.1By1.1d.det_positions.c* is the code which simulates a 1-D camera. It differs from the other 1D-camera-simulating codes mainly in the aspect that in this code the flux of the source (to be simulated) is specified as a command line argument, and the code calculates the value of number of photons to be detected, (in order to get that flux), based on the number of open slits avaliable for the location of the source specified; all the other simulation codes directly accept the number of photons to be generated. This method (of specifying the value of the flux of the source during simulation), is helpful, when multiple sources have to be simulated. Now the bin files can simply be added, without having got to scale the bin-counts (as it is done for the other case). The output detector positions file is stored in the directory, *DET_POS*.

2.3.11 Chk.create_bins.c, Chk.x_correlate.c

To get the detector bins-file, the code, *Chk.create_bins.c* can be used, which is a modified form of *create_bins.c*. [*create_bins.c* used for the 2-D case, can also be used here, with out giving the "-w" switch as one of the command line arguments]. *Chk.create_bins.c* reads the specified simulated detector positions file from the directory, *DET_POS*, and writes the output bin-file in the directory, *DET_BINS*. The code, *create_det_array.c* (used in 2-D case), can be used even here, for getting the detector arrays from the detector bin-file generated. It searches for the given bin-file in the directory, *DET_BINS*, and writes the detector arrays in the directory, *DET_ARRAYS*, in the files named, *det##*, where ## represents the sky element number. The cross-correlation is accomplished by the code, *Chk.x_correlate.c*, which is again a modified form of *x_correlate.c*. It is required that the mask pattern number used during simulations, be specified (as a command line argument), so that the code uses appropriate set of reconstruction arrays. The code uses the detector arrays present in the directory, *DET_ARRAYS* [so it becomes user's responsibility that detector arrays produced from required bin-files are present in *DET_ARRAYS*]. It appends *_pat#* (where # will be the mask pattern number specified), to the name of the output file specified and stores that file in the directory, *PLOTS*.

2.3.12 1By1.c

The code *1By1.c*, is the one which implements the procedure of IROS for the 1-D case. It receives the name of the image file as one of the input arguments, on which IROS has to be employed. [The image file is assumed to be present in the directory, *PLOTS*]. The other argument to this code is the value of the percentage of improvement in RMS value of the image expected, in order to declare that a peak can be considered as a candidate for the next valid source. The code expects that the name of the image file (given as the input argument), has the information about mask pattern number used in the form of "*_pat#*" (where # represents the mask pattern number). This information about mask pattern is required to determine the Point Spread Function file to be used. Once an estimate of a peak's location and strength is made, that source is modelled using the appropriate Point Spread Function. And this modelled response due to one source is removed from the sky response. As has already been described for 2-D case (in *locate_srcs_1By1.c*), here also, the change in RMS value due to removal of all the candidate-peaks are considered, and the one which gives maximum improvement in RMS (when removed) is considered to be a valid source. Again, a structure, *List*, (pointed to by *lp*), is used to hold the information about each of the candidate-peaks and difference in RMS-value due to their removal. RMS value (before and after removal) of the image file is calculated using the procedure, *rms*() (present in the file *1By1.c* itself). Also, the change in RMS due to the removal of first-time-negative peaks are not noted down (in the structure *List*) at all, and hence they are not even candidates for the next source. And after removing each of the sources, the residues that may crop-up at all the previously detected source-locations are checked. If these residues cross certain pre-specified limit (which is a macro, *THRESHOLD*, #define'd to 5, in revision, 1.4), they are modelled and removed from the sky response without checking for the RMS improvement (due to their removal). The information about the sources (and their residues) removed are stored in arrays, *log_elem*[] and *log_peak*[], and the flux that remains at all the sky elements after a source's (or one of its residual's) removal are written in to files in the directory *TRASH*. The names of these files are constructed by appending *_v#*, (where # is the version number), to the image file name given (as the input argument). This procedure is carried on untill there is no peak left which is capable of improving the RMS value, or all the peaks that were picked were first-time-negatives. After the entire process of IROS is completed, using the information in the arrays, *log_elem*[] and *log_peak*[], the consolidated list of sources detected is prepared, and stored in a file, whose name will have, *_perc#.####_SOURCES* appened to the image file name given in the command line (here, #.#### is the value of the percentage of improvement in RMS value expected, which is an other argument to the code). The information about each step of iteration present in the arrays, *log_elem*[] and *log_peak*[] are stored in file whose name is obtained by appending, *_perc#.####_DETAILS* to the input image file-name (containing raw data).

2.3.13 1By1.Interactive.c

There is an interactive version of this 1-D IROS code, called, *1By1.Interactive.c*. It shows the effect due to the removal of every candidate peak, using the *gnuplot*. After every scan, it lists all the candidates picked up, with the difference in RMS found by removing them; Suggests the user to select the candidate-peak giving maximum difference in RMS as the next valid source. The user also can make a choice among the candidates. To get a feel of how *locate_srcs_1By1.c* works, it is suggested to run *1By1.Interactive.c* once.

2.3.14 Chk.1d.add_noise.c

If noise has to be added during the simulations, the code, *Chk.1d.add_noise.c* has to be used. The noise file that has to be added, the scaling factor by which the RMS of the noise file has to be scaled, (which will originally be unity), and the bin file name to which noise has to be added, are the three arguments to be given to this code. The noise files are any of *noise1* to *noise8*, present in the directory, *NOISE* (and the code searches for the noise file specified in this directory). The bin file is searched for in the directory, *DET_BINS*, and the output bin file (input bin file plus noise) is also stored in the same directory.

2.3.15 1By1.1shot.c, add_bins_1By1.1d.c, 1By1.1shot_WithBins.c

The code, *1By1.1shot.c* is an all-in-one code, handling the simulation, image reconstruction and the IROS procedure for the 1-D case. This is accomplished by invoking appropriate executables using the function, *system*(). If simulation has to be carried out in such a way that multiple sources are present in the raw data, the detector positions files due to each of the sources have to be obtained separately, and the corresponding bin-files have to be added. The code, *add_bins_1By1.1d.c* is used for adding the bin files. Since the simulation code, (*Chk.1By1.1d.det_positions.c*) is capable of simulating sources with specified strengths, the code adding bins will have to simply add the bin-counts (without scaling them as has been done in all other bins-adding codes). For this case of multiple sources, the code, *1By1.1shot_WithBins.c* can be used; If the names of the bin files are specified, rest of the things, that is the image reconstruction and the IROS procedures are handled by the code. [This code can as well be used for a case with single source, if its bin-file is avaiable.]

2.3.16 Chk.det_positions.c

Most of the codes written to find out the tolerance limits for a few non-ideal cases are present in this directory, *TWO_DEE*. The code, *Chk.det_positions.c* simulates a 1-D camera whose mask-slit-widths are uneven [if the working of the camera with this non-ideality (of uneven slit-widths) is checked for one mask pattern, it should equally apply for all other patterns]. The code reads the positions of the slit-boundaries (along X direction), [instead of the slits themselves] and randomly (using random number generator, in the file *ran_num_func.c*) introduces the error to these boundaries [the percentage of error to be added to the slit-boundaries is an input argument]. The rest of the procedure in the simulation is same, that is again the random number generator is used to simulate photon strike on the mask plate (which now has uneven slit-widths), and if the position where the photon strikes the mask plate coincides with an open slit, it is "allowed" to pass through and strike the detector (at the specified source angle). The positions on the detector where the photons strike it, are recorded in a file and that is stored in the directory, *DET_POS*. The name of the output file is framed in the format: *Chk_E<#.###>_det_pos_<SigmaX>_<AngleX>* , where *<#.###>* is the percentage error introduced in the slit width, (used up to three decimal places).

2.3.17 Chk.1shot.c, Chk.locate_srcs.c

The code, *Chk.create_bins* can be used for getting the detector bin files, *create_det_array* can be used to generate the detector arrays, and *Chk.x_correlate* can be used to perform the cross correlation. The IROS code, *1By1.c* can be used, for detecting the sources. The code, *Chk.1shot.c* is the all-in-one code handling this non-ideality (of uneven mask slit widths), which invokes all the necessary codes listed above, (except for *1By1.c*) using the function *system*(). For IROS, it uses the code, *Chk.locate_srcs.c*; and the idea in this code has been adopted from the first IROS code for 2-D case, *locate_srcs.c*, that the highest peak will be the next valid source [if the strongest peak is lesser than a threshold (which is a macro, *THRESHOLD* #define'd to 5 in revision, 1.4), the iteration is stopped]. So *Chk.locate_srcs.c* differs from *1By1.c*, mainly in the aspect that it does not consider the RMS criterion to look for the sources. The other difference is in the way in which the files are stored. Both of them use PSF-removal while modelling and removing a source. For any specified source in the simulation, *Chk.1shot.c* introduces errors (in mask slit widths) of percentage values, 0.0(means no error), 0.05, 0.1, 0.2, 0.3,......, and 1.0. And for each of these errors all the sources are removed using *Chk.locate_srcs.c*. A few cases with different sources were executed, and the source-picking was found to degrade by an error of 0.1% itself (that is 0.1 mm approximately).

[*Chk.1shot.c* frames the names of bin files as *bin_Chk_E<#.###>_<SigmaX>_<AngleX>* and the format of the names of image files (one in which *Chk.x_correlate* stores the output) will be *p_Chk_E<#.###>_<SigmaX>_<AngleX>_pat<PatternNumberUsed>*]

2.3.18 Chk.mask.tilt.det_positions.c

The code, *Chk.mask.tilt.det_positions.c* simulates a 2-D camera in which the plane of the mask plate is tilted by some angle with the detector plane. To realise the same effect, in the simulation code, the plane of the detector wires is tilted with respect to the plane of the mask plate (along both X and Y directions). This is just to ensure that the reference with which the sky elements were calculated is not altered (by tilting the mask-plane itself). And now the distance between the mask plate and the detector wires will be a function of the tilt angle. The code receives the values of the angles by which the mask plate will have to be tilted (in the code actually the detector is tilted) along the X and along Y directions (θ_x and θ_y). And the following terms μ_x and μ_y are calculated.

$$\mu_x = \frac{1}{\cos\theta_x + \tan\theta_x * \sin\theta_x}$$
$$\mu_y = \frac{1}{\cos\theta_y + \tan\theta_y * \sin\theta_y}$$

And for every photon strike on the detector, the new X and Y positions (if the detector were to be tilted at the given angles), are calculated using,

$$X_{WithTilt} = X_{WithOutTilt} * \mu_x$$
$$Y_{WithTilt} = Y_{WithOutTilt} * \mu_y$$

These new readings are recorded in a file, which is stored in the directory, *DET_POS*. The name of the output file is framed as,
Chk.mt_<TiltAngleX>_<TiltAngleY>_det_pos_<SigmaX>_<SigmaY>_<AngleX>_<AngleY>.

2.3.19 Chk.mt.1shot.c

And rest of the steps in image reconstruction and IROS remain the same. That is, the codes, *wire_positions.c*, *create_bins.c*, *create_det_array.c*, *x_correlate.c*, *images_y.c*, *add_noise.c* and *locate_srcs.c* can be used. All these codes have been used in the all-in-one code, *Chk.mt.1shot.c*. If multiple sources have to be simulated, the detector positions where the photons strike have to be obtained for each of the sources separately (by executing *Chk.mask.tilt.det_positions.c*), and the corresponding bin files have to be generated (using *create_bins.c*). Then these bin files have to be added (wire-by-wire), after scaling the bin-counts, depending on the relative strengths of the sources expected. This can be accomplished by the code, *add_bins.c*; two things have to be told about any source (whose bin files is going to be added to bin files of some other source) in the command line argument to this code. One is the name of the bin file (with out the wire part, *_wire#*, where # can be 1, 2, 3,... or 8) and the other is the scaling factor by which the bin counts (collected from one source, on all the eight wires) have to be scaled (in all the eight bin files). The name of the output file will also have to be specified, and the code searches for the input bin files in the directory, *DET_BINS*, and the output file is also stored in the same directory.

2.3.20 1shot_with_bins.c, 1shot_with_bins_NoiseFactor.c

The all-in-one code to be used for this new set of (eight summed) bin files having information about multiple number of sources, is *1shot_with_bins.c*. This uses the codes, *create_det_array.c*, *x_correlate.c*, *images_y.c*, *add_noise.c* and *locate_srcs.c* for handling rest of the steps. Instead of *1shot_with_bins.c*, *1shot_with_bins_NoiseFactor.c* can be used, which differs from *1shot_with_bins.c* only in using *locate_srcs_NoiseFactor.c* for IROS, instead of *locate_srcs.c*. For this case of non-ideality (of tilted mask plane with respect to the detector plane), the simulation results showed that the tolerance limit for tilt in mask plate along X direction is 2° and that along Y direction is 4°.

[The all-in-one codes used for this case of non-ideality frame the name of the bin file as,
bin_Chk.mt_<θ_x*>_<*θ_y*>_<SigmaX>_<SigmaY>_<AngleX>_<AngleY>_wire<WireNum>* and the format of the name of the corresponding image file will be
p_Chk.mt_<θ_x*>_<*θ_y*>_<SigmaX>_<SigmaY>_<AngleX>_<AngleY>_wire<WireNum>*.]

2.3.21 Chk.wire.tilt.det_positions.c

The code, *Chk.wire.tilt.det_positions.c* is the simulation code which simulates a camera whose mask slits are tilted to an angle with respect to the wires. This handles only the case when *all* the mask slits are tilted at the *same* angle. This case has been suitably modified as wires being tilted with respect to the wires. The only change (when compared to the original simulation code *det_positions.c*) is that the position of the photon strike on the detector will have to be shifted depending on the tilt-angle by which the wires would have got oriented and then decide which wire the photon will be recorded by. The new co-ordinates of the photon strike on the detector will be,

$$X_{WithTilt} = X_{WithOutTilt} * \cos\theta_{tilt} + Y_{WithOutTilt} * \sin\theta_{tilt}$$
$$Y_{WithTilt} = Y_{WithOutTilt} * \cos\theta_{tilt} + X_{WithOutTilt} * \sin\theta_{tilt}$$

The code needs the tilt angle to be specified by which the mask slits will be oriented with the wires. [The code tilts the wires with respect to the mask slits]. Again the output file is stored in the directory, *DET_POS*. The name of the output file is framed as, *Chk.wt_*<θ_{tilt}>*_det_pos_*<*SigmaX*>_<*SigmaY*>_<*AngleX*>_<*AngleY*> .

2.3.22 Chk.wt.1shot.c, Chk.wt.1shot_NoiseFactor.c

And for rest of the steps, the codes, *wire_positions.c*, *create_bins.c*, *create_det_array.c*, *x_correlate.c*, *images_y.c*, *add_noise.c*, *locate_srcs.c* can be used, and these have actually been used in the all-in-one code, *Chk.wt.1shot.c*,which handles this non-ideality of mask slits becoming tilted to an angle with respect to the wires. *Chk.wt.1shot_NoiseFactor.c* is another all-in-one code handling the same non-ideality (of mask slits not being perpendicular to the detector wires). The main difference when compared to *Chk.wt.1shot.c* is that, *Chk.wt.1shot_NoiseFactor.c* uses *locate_srcs_NoiseFactor.c* instead of *locate_srcs.c*.

Again, for the case when multiple sources have to be simulated, a similar procedure, as described for the non-ideality of mask plane tilting to an angle with respect to the detector plane, will have to be followed. [And again the all-in-one code to be used will be either of *1shot_with_bins.c* or *1shot_with_bins_NoiseFactor.c*].

[The all-in-one-codes used frame the name of the bin file as, *bin_Chk.wt_*<θ_{tilt}>_<*SigmaX*>_<*SigmaY*>_<*AngleX*>_<*AngleY*>*_wire*<*WireNum*> and the names of the corresponding image files are framed as *p_Chk.wt_*<θ_{tilt}>_<*SigmaX*>_<*SigmaY*>_<*AngleX*>_<*AngleY*>*_wire*<*WireNum*>]

2.3.23 Chk.mask.width.det_positions.c

The other issue which had to be dealt with was the mask plate width. Till this point, all the simulation codes simply neglected the mask plate width. But it was found by ISAC that for stability, the mask plate should be atleast 2 mm thick, and the effect of this finite width had to be checked using the simulations. The simulation code, *Chk.mask.width.det_positions.c* simulates a camera whose mask plate width can be specified. The only extra step (when compared to *det_positions.c*) is that for every incident photon, which strikes the mask plate such that, that position at which it has fallen on the mask plate coincides with an open slit, that photon should then be made to travel towards the detector, in to the slit, at the given source angle. If it so happens that while/after traversing a vertical distance equal to the height (depth) of the slit (which is specified as 2 mm), the photon hits the inner wall of that slit, that particular photon will have to be neglected. And if in case, the photon appears at the open end of the slit (towards the inner surface of the mask plate), it will be allowed to travel further to strike the detector, at the source angle given, and that photon position is recorded. As is very clear, the only effect due to this finite mask plate thickness is, reduction in the number of photons received from any source, and an increase in the Poisson noise. The photons strike positions on the detector are recorded in a file whose name is framed as *Chk.mpw_*<*MaskPlateWidth*>*_det_pos_*<*SigmaX*>_<*SigmaY*>_<*AngleX*>_<*AngleY*> . And this file is stored in the directory, *DET_POS*.

2.3.24 Chk.mpw.1shot_NoiseFactor.c, Chk.mpw.1shot_RMS.c, Chk.mpw.1shot_1By1.c

Again, the other codes which have to be used for rest of the steps are, *wire_positions.c*, *create_bins.c*, *create_det_array.c*, *x_correlate.c*, *images_y_RMS.c* and *add_noise.c*. There are three all-in-one codes which have been written for this case of finite mask plate width, *Chk.mpw.1shot_NoiseFactor.c*, which uses the code *locate_srcs_NoiseFactor.c* for IROS, *Chk.mpw.1shot_RMS.c*, which uses the code *locate_srcs_RMS.c* for IROS and *Chk.mpw.1shot_1By1.c*, which uses *locate_srcs_1By1.c* for IROS.

[The bin file names are framed as
bin_Chk.mpw_<MaskPlateWidth>_<SigmaX>_<SigmaY>_<AngleX>_<AngleY>_wire<WireNum>
and the image file will be
p_Chk.mpw_<MaskPlateWidth>_<SigmaX>_<SigmaY>_<AngleX>_<AngleY>_wire<WireNum>]

2.3.25 find_peaks_func_NoiseFactor.c, find_peaks_func_RMS.c, CheckFalseAlarm_1By1_func.c

There are a few versions of the code finding the peaks among the 55 image files. Till now only two, *find_peaks_func.c* and *find_peaks_func_1By1.c* have been mentioned. The subroutine, *find_peaks_func_NoiseFactor.c* differs from *find_peaks_func.c* in two ways; it compares the absolute values of the peaks rather than the peaks themselves (as in *find_peaks_func.c*), and it does not check for false alarms. The peak obtained is simply put in the handles provided by the calling procedure (in the form of arguments to the procedure). The procedure, *find_peaks_func_RMS.c*, searches for the peaks only in the specified range of sky elements. It checks for false alarms and makes a note of it in a data structure [and even if any (highest) peak is discovered to be a false alarm, it is returned to the calling procedure]. When the iteration is going to be aborted, the calling (main) procedure calls *find_peaks_func_RMS.c*, with the last argument as a valid file pointer [in all other invokations this argument is *NULL*]. On noticing a valid file pointer, this procedure, puts all the information collected about the false alarms till that point in the iteration, in the file whose file pointer is provided. The procedure in *find_peaks_func_1By1.c* does similar things as what *find_peaks_func_RMS.c* does, except that it's mainly memory operations (using arrays) instead of file operations (as in *find_peaks_func_RMS.c*); and one more important distinction is that *find_peaks_func_1By1.c* does not handle the false alarm issue. A separate procedure in the file, *CheckFalseAlarm_1By1_func.c*, which also uses memory opeartions checks for false alarms. The issue of false alarms has to be discussed, and so to be able to modify it independently (if necessity arises), a separate procedure was opted. [*CheckFalseAlarm_1By1_func.c* is used by the IROS code, *locate_srcs_1By1.c*.]

2.3.26 Coll_det.positions.c, convolution.c, get_angles_y.c, get_tables.c, shadow_files.c, plot_psf.c

There are a few assorted codes like the simulation code *Coll_det.positions.c*, which differs from the code, *det.positions.c*, only in the aspect that it prints X and Y positions of the photon strike on the detector, skipping the step of judging which wire-cell (and hence the wire) the Y-positions belong to. This can only be used to plot the output detector-positions file. One other code, *convolution.c* can be used to convolve a 126*55 matrix. The convolution algorithm followed is based on the assumption that, the sources seen would have got distributed among adjacent sky locations, following a Gaussian distribution. In the codes like *locate_srcs.c*, the convolution feature has not been incorporated, and to get the convolved form of the output matrices, [MATRIX and ORG-MATRIX], this code can be used. The code, *get_angles_y.c* prints the values of the angles corresponding to the sky elements along the Y direction on the standard output. [These values have been stored in the file, *angles_y.data*]. The code, *get_tables.c* generates a table of information for each of the mask patterns. The table will contain the number of open slits, number of closed ones and total number of slits (getting illuminated) corresponding to all the sky elements (along X direction). These files (named as *pattern#*, where # will be the mask pattern number), will be stored in the directory, *TABLES*. [A simple awk (*add_angles.awk*) script was used to insert the angles corresponding to each of the sky elements as one of the columns in these files]. The code, *shadow_files.c* generates all the shadow files corresponding to all the six mask patterns. The shadow files are stored in the directories *SHADOW#* (# standing for the mask pattern number), and each of the files are named as *sha*@@ (where @@ represents the sky element(X) number the shadow file corresponds to). The code *plot_psf.c* produces the Point Spread Functions for each of the six mask patterns. The output files have been stored in sub-directories, *PSF_PATTERN#* (where # represents the mask pattern number), under the directory, *PLOTS* and the files are named, *p_PSF@@*, where @@ represents the sky element number along the X direction. It is these PSF files and shadows files which are used while "removing" a source during IROS step (by the codes implementing the IROS procedure).

2.3.27 ASI_gen_aperture_func.c, ASI_x_correlate.c

The code, *ASI_gen_aperture_func.c* generates the mask aperture functions and complementary mask aperture functions (the C_k^j's and $\bar{C}_k^j$'s defined in the first chapter *INTRODUCTION*, of this report) for all the six mask patterns. The aperture functions are stored in sub-directories, *pattern#* (# representating the mask pattern number), under the directory *APERTURE_FUNC*. The files are simply named 1,2,3....125 after the sky elements (X) they correspond to. Similarly the complementary aperture functions are stored in sub-directories called *comp_pattern#* (# representating the mask pattern number), under the directory *APERTURE_FUNC*. The files are again named after the sky elements. The code *ASI_x_correlate.c* is used to illutrate the reconstruction procedure. First, the source component is obtained by correlating the detector array with the aperture function, then the background component is obtained by correlating the complementary aperture function (with the detector array), and finally the second result is subtracted from the first one and that gives the required image file. The code puts the result of correlation performed with the aperture functions in the file under the directory *PLOTS*; the name of the file is obtained by appending *_pat#_asis* (# specifies the mask pattern used during simulations) to the output file name specified (as an argument to the code). The name of the file (stored in *PLOTS*) containing correlation result obtained by using the complementary aperture functions is framed by appending *_pat#_comp* (again # stands for the mask pattern used) to the output file name specified . Then the final image obtained after subtraction is named by appending *_pat#_sub* to the output file name specified. This image should be same as the one obtained by following the procedure which uses Reconstruction Arrays during cross correlation. Before the execution of this code, it should be ensured that the detector arrays, files in *DET_ARRAYS*, have the appropriate data. (or in other words, the execution of *ASI _x_correlate.c* should follow that of any code which produces the detector arrays).