Derivation of gaintable using Kuhn et al. (1991)'s principle

It is assumed that the same object o_{ij} (i=0,..,n_x-1, j=0,.., n_y-1) is imaged several times with different relative offsets. Note that i and j are detector pixel coordinates of the image. The k-th observed image a^k_{ij} may be expressed as follows:

$$a^k_{ij} = c_k o_{i-l_k,j-m_k} f_{ij}. \tag{1}$$

Here l_k and m_k are the offsets of the a^k_{ij} determined with respect to o_{ij}. The parameter f_{ij} is the detector gain pattern, which is assumed to be constant over time, and the parameter c_k represents the detector-postion independent overall gain change. The logarithmic form of the above equation is

$$A^k_{ij} = C_k + O_{i-l_k,j-m_k} + F_{ij}. \tag{2}$$

where the capital letters represent the logarithmics of the corresponding variables. The problem is then defined as follows: "Determine C_k, F_{ij}, and O_{ij} when A^k_{ij} are given. "

The formal solutions for C_k and F_{ij} can be found by optimizing the functional

$$H = \sum_k \sum_{ij} (C_k + O_{i-l_k,j-m_k} + F_{ij} - A^k_{ij})^2 p(i-l_k) q(j-m_k). \tag{3}$$

The two functions p and q are introduced to deal with the boundary effects, and are defined as

$$p(i) = \begin{cases} 1 & : \quad 0 \le i \le n_x - 1 \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4}$$

and

$$q(j) = \begin{cases} 1 & : \quad 0 \le j \le n_y - 1 \\ 0 & : \quad \text{otherwise} \end{cases} \tag{5}$$

From the condition of $\partial H / \partial F_{ij} = 0$, we have

$$F_{ij} = \frac{\sum_k (-C_k - O_{i-l_k,j-m_k} + A^k_{ij}) p(i-l_k) q(j-m_k)}{\sum_k p(i-l_k) q(j-m_k)} \tag{6}$$

and from the condition of $\partial H / \partial C_k = 0$, we have

$$C_k = \frac{\sum_{ij} (-F_{ij} - O_{i-l_k,j-m_k} + A^k_{ij}) p(i-l_k) q(j-m_k)}{\sum_{ij} p(i-l_k) q(j-m_k)} \tag{7}$$

For the determination of O_{ij}, we optimize the functional

$$H = \sum_k \sum_{ij} (C_k + O_{ij} + F_{i+l_k,j+m_k} - A^k_{i+l_k,j+m_k})^2 p(i+l_k) q(j+m_k) \quad (8)$$

with respect ot O_{ij} and find the formal solution

$$O_{ij} = \frac{\sum_k (-C_k - F_{i+l_k,j+m_k} + A^k_{i+l_k,j+m_k}) p(i+l_k) q(j+m_k)}{\sum_k p(i+l_k) q(j+m_k)} \quad (9)$$

Note that Equations (6), (7), and (9) are only formal since they are still related to among another. Thus the true solution can be obtained only after iterations. An implementation of this principle is given below.

```
function gaincalib_new, images, x0, y0, object=object,  maxiter=maxiter
;+
;  NAME:  GAINCALIB_NEW
;  PURPOSE:    Produce either a gain table or an offset table
;              from a set of images
;              taken shifted with respect to one another
;  CALIING SEQUENCE:
;          Result = gaincalib(images, additive=additive)
; INPUT:
;          images    a three-dimensional array representing
;                    a sequence of two-dimensional images
;                    images(*,*,k) ( k=0, 1,.., N-1).
;                    It is strongly recommended that the number
;                    of elements in each image is a power of 2
;                    for the efficent FFT processing.
;          x0        an array of x-shift
;          y0        an array of y-shift
;
; OUTPUT:
;        Result     the gain table if the keyword  ADDITIVE is not set
;                   or the offset table if the keyword is set.
; INPUT KEYWORDS:
;        maxiter   maximum # of iternation (default=5)
; OUTPUT KEYWORD:
;        object      flat-field corrected objecto
;-
```

```
if n_elements(maxiter) eq 0 then maxiter=5
if n_elements(iflat) eq 0 then if keyword_set(additive) then $
iflat=0. else iflat=1.
if keyword_set(additive) then additive=1 else additive=0
s=size(images)
i = indgen(s(1))#replicate(1, s(2))
j = replicate(1, s(1))#indgen(s(2))

       ; Initial Estimates

 flat = total(images, 3)/s(3)
 logimages = alog(images>(0.01*median(flat)))
 Flat = alog(flat) & Flat = Flat-median(Flat)


 C = fltarr(s(3))
 w = fltarr(s(3))
 for k=0, s(3)-1 do begin
C(k) = median(logimages(*,*,k)-flat)
endfor
 C = C -C(0)

      ; Iteration

object = fltarr(s(1), s(2), s(3)-1)

for iter=0, maxiter do begin
flat0 = flat
   tmp=0.
   a=0. & b =0.
     aa=0.0 & bb=0.0
    for k=0, s(3)-1 do begin
    weight = (i ge -x0(k)) and (i le -x0(k)+s(1)-1) $
               and (j ge -y0(k)) and (j le -y0(k)+s(2)-1)
    aa = aa + (-C(k) - $
       shift_sub(Flat, -x0(k), -y0(k)) $
      +shift_sub(logimages(*,*,k),  -x0(k) , -y0(k)))*weight
    bb = bb+weight
    endfor
```

```
   Object = aa/(bb+0.001)
   a=0. & b=0.0
    for k=0, s(3)-1 do begin
    weight = (i ge (x0(k)) ) and (i le x0(k)+s(1)-1) $
               and (j ge (y0(k))) and (j le y0(k)+s(2)-1)
    ob = shift_sub(Object,  x0(k), y0(k))
    C(k) = total((-Flat -ob+logimages(*,*,k))*weight)/total(weight)
    a = a + (-C(k) - ob+logimages(*,*,k))*weight
    b = b+weight
     endfor
flat = a/(b+0.001)
error = stdev(flat-flat0)
    print, 'iteration #  =', iter , '  error=', error
if error le 1.0e-5 then goto, next
endfor
next:
Object = exp(Object+median(flat))
Flat = exp(flat-median(flat))
c = exp(c)
return, flat
end
```