Image Alignment

Jongchul Chae
Big Bear Solar Observatory, NJIT

May 18, 1999

1. Introduction

Image alignment is crucial for the analysis of different kinds of data obtained from joint observations. It is basically a linear mapping which consists of translation, scaling, rotation and flipping. It assumes the existence of a reference image. One images are aligned, the aligning information should be kept for later use. There may be two approaches of keeping aligning information. One is to map each image in the image coordinates of the reference image, and save it into a file for later use. This is simple to use, but requires disk space for saving aligned images. The other is to keep only the basic mapping parameters. In this case, the mapping should be made whenever image alignments are necessarey. We choose the latter approach. This approach requires non-ambiguous and general definition of mapping parameters, and mapping relation.

2. Image Alignment as a Mapping

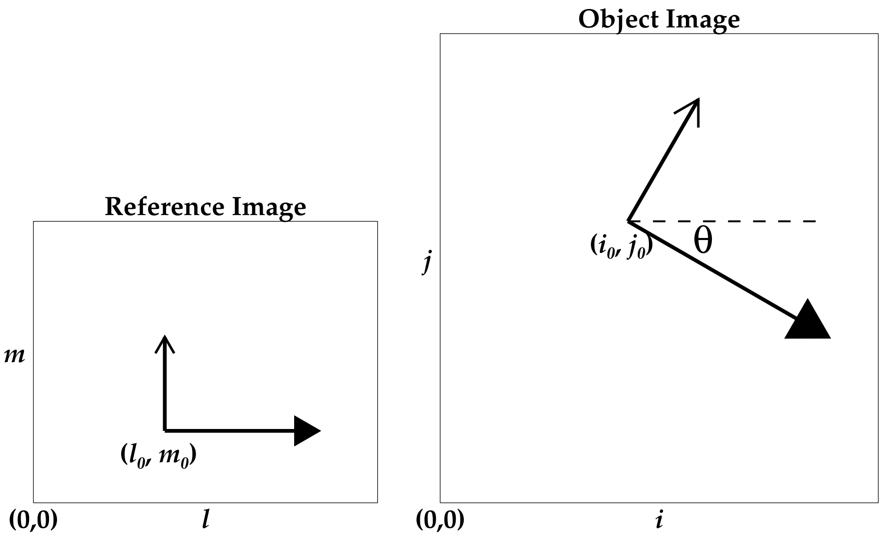

Fig. 1.— Illustration of image alignment.

Translation, Rotation, and Scaling Let's consider a relatively simple case of image alignment, in which case translation, rotation, and scalling should be considered. Figure 1 illustrates this situation. The reference and object images are denoted by $R(l, m)$ and $O(i, j)$. Then aligning the object image to the reference image is reduced to determine a new image $Q(l, m)$ which contains the same information as O, but has the same image coordinates as the reference image. Once the basic mapping parameters and mapping relation are known, image aligning is done in two steps.

1. For the every pair of l (l=0,.., N_l-1) and m (m=0,..,N_m-1), determine the corresponding pair of i=$i(l,m)$, j=$j(l,m)$.
2. Create the new image $Q(l,m)$ by taking the interpolated values of O at the positions $(i(l,m), j(l,m))$.

In the situation depicted in Figure 1, the mapping relation can written in the way:

$$s(i - i_0) = \quad (l - l_0)\cos\theta + (m - m_0)\sin\theta \tag{1}$$
$$s(j - j_0) = \quad -(l - l_0)\sin\theta + (m - m_0)\cos\theta \tag{2}$$

in which we introduced the mapping parameters: l_0, m_0, i_0, j_0, θ, and s. Here s is the scalling parameter which is defined by the ratio of the physical size of an object image pixel, to that of a reference image pixel. The parameter θ is the counterclockwise angle of the horizontal ("i") direction of the object image frame measured with respect to the horizontal("l") direction of the reference image frame (in radians). Note that l_0 and m_0 are free parameters, others should be determined by comparing the two images.

Non-square pixel Images Either the object image or the reference image has non-square pixels. We introduce two more parameters: r, the ratio of the physical size of an object image pixel in the j-direction to that in the i-direction, and q, the ratio of the physical size of a reference image pixel in the m-direction to that in the l-direction. Then the mapping relation should be generalized to:

$$s(i - i_0) = \quad (l - l_0)\cos\theta + q(m - m_0)\sin\theta \tag{3}$$
$$sr(j - j_0) = \quad -(l - l_0)\sin\theta + q(m - m_0)\cos\theta. \tag{4}$$

Now the exact definition of the parameter s is the ratio of the physical size of an object image pixel measured in the i-direction, to that of a reference image pixel measured in the l-direction.

Flipping We introduce the flipping parameters f and g, which should have absolute values of unity, and rewrite the above equations:

$$fs(i - i_0) = \quad (l - l_0)\cos\theta + q(m - m_0)\sin\theta \tag{5}$$
$$gsr(j - j_0) = \quad -(l - l_0)\sin\theta + q(m - m_0)\cos\theta. \tag{6}$$

Using these relations, the flipping in the i-direction can be expressed by choosing f=-1, and the flipping in the j-direction is done with $h = -1$.

IDL structure for image aligning . The image alignment of an image with respect to the reference image is defined by specifying 1) the image to be alinged, and the reference, and 2) the 10 parameters: l_0, m_0, i_0, j_0, q, r, θ, s, f, and g. Therefore it is convenient to introduce an IDL structure which has these things as fields. The following IDL routine is to define the basic structure.

IDL routine for image aligning Equations (5) and (6) define the mapping which is required to align the object image to the reference image. Symbolically, the mapping can be represented in a way like

$$(l, m) \Longrightarrow (i, j). \tag{7}$$

Once the mapping structure is known, the image coalignment is achieved in IDL by using the routine `alignmap`.

Successive Mapping Sometimes the object image $O(i,j)$ can be the reference image of the another image, for example, $T(i',j')$. Then the alignment of $T(i',j')$ with respect to $R(l,m)$, $(l,m) \Longrightarrow (i',j')$, is a succession of two mappings:

$$(l,m) \Longrightarrow (i,j) \Longrightarrow (i',j') \tag{8}$$

This kind of successive mapping can be handled by defining a new set of parameters which combine the two set of mapping parameters. Note that the order of the mappings is important. Let's denote the second mapping parameters by the varaibles with the prime superscript, and the combined mapping parameters by the ones with the double prime subscript. Then it follows: q''=q, r''=r', θ''=θ+$fg\theta'$, s''=ss', $f'' = ff'$, and g''=gg'. The expressions for i_0'' and j_0'' are somewhat complicated. If we choose l_0''=l_0, m_0''=m_0, then it follows:

$$i_0'' = i_0' + \frac{(i_0 - l_0')\cos\theta' + q'(j_0 - m_0')\sin\theta'}{f's'} \tag{9}$$

$$j_0'' = j_0' + \frac{-(i_0 - l_0')\sin\theta' + q'(j_0 - m_0')\cos\theta'}{g's'r'} \tag{10}$$

It should be remembered that this kind of sucessive mapping is meaningful only when the object image in the first mapping becomes the reference image in the second mapping. Thus it should be guarranteed that q'=r. The IDL routine `alignmul` does the muliplication of the structures for the successive mapping.

Inversion The inversion mapping: $(i,j) \Longrightarrow (l,m)$ is defined by defining a new set of parameters: q'=r, r'=q, s'=$1/s$, i_0'=l_0, j_0'=m_0, l_0'=i_0, m_0'=j_0, and θ'=-$fg\theta$. The IDL routine `aligninv` does this operation.

3. How to determine alignment parameters

There is no general way to determine the alignment parameters. A good way is to decompose the mapping into a succession of basic mappings. Then one needs to know how to determine the alignment parameters of the basic mappings. Let's examine how to determine the alignment parameters in several basic mappings.

Translation Image alignments of homogeneous images obtained by the same instrument are often reduced to image tranlation. In this case, we have $s = 1$ and $\theta = 0$. The value of $r = q$ is usually known in advance, and is set to be 1 if the images have square pixels. The parameters l_0 and m_0 can be freely chosen. Then what to be determined is the two translation parameters: i_0 and j_0. The values of $i_0 - l_0$ and $j_0 - m_0$ are then the offsets of the object image with respect to the reference image. A very popular way of determining these offsets is to maximize the cross-correlation of the two images over the shift, using the FFT algorithm.

4. IDL Routines

- `aligndiv`: Divison Operation of two alignment structures
- `aligninv`: Inversion operation of an aligment structure
- `alignmap`: Determine a new image aligned to the reference image
- `alignmul`: Multiple operation of two alignment structures
- `alignoffset`: Determine the offsets of an image witht respect to reference
- `alignrdiv`: Reverse division operation of two alignment structures
- `alignstr`: Initialize the alignment structure

;

```
function alignstr, dummy
;+
; NAME: alignstr
; PURPOSE: Initialize the alignment structure
; CALLING SEQUENCE:
;         a = alignstr()
; EXPLANATION:
;  a.object: a string, the information on the object image to be aligned
;          (for example, a file name)
;  a.reference: a string, the information on the reference image
;  a.ref_origin: a two-element float array,
;           the origin in the reference image coordinates
;  a.obj_origin:  a two-element float array,
;           the origin in the object image coordinates
;  a.rotation: a float scalar, the counterclockwise
;           angle of the  horizontal ("x") direction of the
;           object image frame measured with respect to
;           the horizontal("x") direction of the reference image frame
;           (in radians)
;  a.ref_aspect: a float scalar, the ratio of the physical size of
;           a reference image pixel measured in the
;           horizontal direction to that measured in the vertical
;           direction
;  a.obj_aspect: a float scalar, the ratio of the physical size of
;           an object image pixel measured in the
;           horizontal direction to that measured in the vertical
;           direction
;  a.scale: a float scalar, the ratio of the physical size of
;           an object image pixel measured in the horizontal direction
;           to that of a reference pixel measured in the vertical
;           direction
```

```
;  a.flip: a two-element array, the flipping parameters which have a value
;          of either 1 or -1.
;-
return, {align,  $
         object:     '', $
         reference:  '', $
         ref_origin: [0.,0.], $
         obj_origin: [0.,0.], $
         rotation:   0.,  $
         ref_aspect: 1.,  $
         obj_aspect: 1.,  $
         scale:      1., $
         flip:       [1,1]  }
end
;

;

function alignmap, object_image, alignpar, xrange, yrange, mag=mag
;+
; NAME: alignmap
; PURPOSE: Determine a new image alingned with the reference image
; CALLING SEQUENCE:
;      New_Image = alignmap(Object_Image, alignpar, xrange, yrange)
; INPUTS:
;  Object_Image  object image to be alingned with the reference image
;  alignpar      alignment parameters, a structure variable
;                which is consitent with the structure
;                defined by alignstr().
;  xrange        the mapping range of x-direction indice of
;                the reference image, a two-element array
;  yrange        the range of y-direction indice, atwo-element array
; KEYWORD INPUT:
;  mag           the magnification factor (default=1)
;-
ns = n_elements(alignpar)
if n_elements(mag) eq 0 then mag=1
l = findgen((xrange(1)-xrange(0)+1)*mag)/mag+xrange(0)
l = l#replicate(1., (yrange(1)-yrange(0)+1)*mag)
m = findgen((yrange(1)-yrange(0)+1)*mag)/mag+yrange(0)
m = replicate(1., (xrange(1)-xrange(0)+1)*mag)#m

l0 = alignpar.ref_origin(0)
m0 = alignpar.ref_origin(1)

i0 = alignpar.obj_origin(0)
```

```
j0 = alignpar.obj_origin(1)

theta = alignpar.rotation
q = alignpar.ref_aspect
r = alignpar.obj_aspect
s = alignpar.scale
f = alignpar.flip(0)
g = alignpar.flip(1)

tmp = (l-l0)*cos(theta) + q*(m-m0)*sin(theta)
i   = tmp/(s*f) + i0

tmp = -(l-l0)*sin(theta) + q*(m-m0)*cos(theta)
j  = tmp/(s*r*g) + j0


New_Image = interpolate(Object_image, i, j)

return, New_Image
end
;

   ;

;+
; NAME:   alignmul
; PURPOSE:  Multiple operation of two alignment structures
; CALIING SEQUENCE:
;   C=alignmul(A, B)
; INPUTS:
;      A   the first alignment structure
;          representing the mapping: (l,m) ---> (i,j)
;      B   the second alignment structue
;          representing the mapping: (i,j) ---> (t,u)
; OUPUT:
;      C  the multiplication of A and B,
;          representing the mapping: (l,m) ---> (t,u)
; REMARK:
;      the order of inputs A and B are important.
function alignmul,  a, b
           ;  Compatability Check
check = a.object eq b.reference
check = check and a.obj_aspect eq b.ref_aspect
if not check then begin
  print, "Incompatible Inputs in ALIGNMUL"
 return, a
```

```
endif
c = alignstr()
c.reference = a.reference
c.ref_aspect = a.ref_aspect
c.object = b.object
c.obj_aspect = b.obj_aspect
c.flip = a.flip*b.flip
c.scale = a.scale*b.scale
c.rotation = a.rotation + a.flip(0)*a.flip(1)*b.rotation
c.ref_origin = a.ref_origin
c.obj_origin(0) = b.obj_origin(0) + $
  ( (a.obj_origin(0)-b.ref_origin(0))*cos(b.rotation) + $
 b.ref_aspect*(a.obj_origin(1)-b.ref_origin(1))*sin (b.rotation)) $
/(b.flip(0)*b.scale)
c.obj_origin(1) = b.obj_origin(1) + $
  (- (a.obj_origin(0)-b.ref_origin(0))*sin(b.rotation) + $
 b.ref_aspect*(a.obj_origin(1)-b.ref_origin(1))*cos (b.rotation)) $
/(b.flip(1)*b.scale*b.obj_aspect)
return, c
end
;

    ;

;+
;  NAME: aligninv
;  PURPOSE: Get an inversion alignment structure
;  CALLING SEQUENCE:
;      B = aligninv(A)
;  INPUT:
;      A an alignment structure, representing the mapping
;           (l,m) ----> (i,j)
;  OUPUT:
;      B the inversion alignment structure, representing
;         the mapping   (i,j) ---> (l,m)
;-

b=alignstr()
b.object = a.reference
b.reference = a.object
b.ref_origin= a.obj_origin
b.obj_origin= a.ref_origin
b.ref_aspect = a.obj_aspect
b.obj_aspect = a.ref_aspect
b.flip = a.flip
b.scale = 1./a.scale
```

```
b.rotation = -a.flip(0)*a.flip(1)*a.rotation
return, b
end
;

;

;+
; NAME:   aligndiv
; PURPOSE:  Division operation of two alignment structures
; CALIING SEQUENCE:
;   C=alignmul(A, B)
; INPUTS:
;      A   the numerator alignment structure
;          representing the mapping: (l,m) ---> (t,u)
;      B   the denominator alignment structue
;          representing the mapping: (i,j) ---> (t,u)
; OUPUT:
;      C  the division of A by B,
;          representing the mapping: (l,m) ---> (i,j)
; REMARK:
;      the order of inputs A and B are important.
function aligndiv,  a, b
           ;  Compatability Check
check = a.object eq b.object
check = check and a.obj_aspect eq b.obj_aspect
if not check then begin
  print, "Incompatible Inputs in ALIGNMUL"
 return, a
endif
c =alignmul(a, aligninv(b))
return, c
end
;

    ;

;+
; NAME:   alignrdiv
; PURPOSE:  Reverse division operation of two alignment structures
; CALIING SEQUENCE:
;   C=alignmul(A, B)
; INPUTS:
;      A   the denominator alignment structure
;          representing the mapping: (l,m) ---> (i,j)
;      B   the numerator alignment structue
```

```
;          representing the mapping: (l,m) ---> (t,u)
; OUPUT:
;      C  the division of B by A,
;          representing the mapping: (i,j) ---> (t,u)
; REMARK:
;      the order of inputs A and B are important.
function alignrdiv,  a, b
            ;  Compatability Check
check = a.reference eq b.reference
check = check and a.ref_aspect eq b.ref_aspect
if not check then begin
  print, "Incompatible Inputs in ALIGNMUL"
 return, a
endif
c =alignmul(aligninv(a), b)
return, c
end
;


   ;


;+
; NAME : alignoffset
; PURPOSE :
;        Determine the offsets of an image with respect to a reference image
; CALIING SEQUENCE
;       OFFSET  =  alignoffset(IMAGE, REFERENCE)
; INPUTS:
;      IMAGE         the object image
;      REFERENCE     the reference image
; OUTPUT:
;     OFFSET         a two-element array of the offset values
;                    defined by  OFFSET  = (i, j) - (l, m)
;                    where (i, j) is the object image coordinates of a feature
;                   and (l, m),  its reference image coordinates.
;
; REMARK:
;            FFT method is used to mximize the cross-correlation
; HISTORY:
;      1999 May,   J. Chae
;-
function alignoffset, image, reference
                                   ;  Check compatibility

si = size(image)
sr=size(reference)
```

```
if not(si(1) eq sr(1) and si(2) eq sr(2)) then begin
 print, 'Incompatbile Images : getoffset'
return, [0,0.]
endif
               ; Maximize cross-correlation over the indeteger pixels
cor = float(fft(fft(image, 1)*fft(reference, -1), -1))

tmp = max(cor, s)
x0 = s(0) mod si(1)
y0 = s(0) / si(1)
               ; Maximize the cross-correlation over the subpixels
cc = (shift(cor, -x0+1, -y0+1))(0:2, 0:2)
x1 = (cc(0,1)-cc(2,1))/(cc(2,1)+cc(0,1)-2.*cc(1,1))*.5
y1 = (cc(1,0)-cc(1,2))/(cc(1,2)+cc(1,0)-2.*cc(1,1))*.5
x = x0+x1
y = y0+y1
return, [x, y]
end
;
```