
Subject: Re: Image registration

Posted by [Markus Schmassmann](#) on Thu, 01 Jun 2017 11:13:42 GMT

[View Forum Message](#) <> [Reply to Message](#)

There are a lot of evil loops in there, as far as I can see none of them necessary.

I eliminated them, but did not verify the code. Once debugged, it should be a lot faster. Markus

<http://www.idlcoyote.com/tips/forloops.html>

<http://www.idlcoyote.com/tips/forloops2.html>

On 06/01/2017 08:44 AM, Klemen wrote:

> Take a look at this links. I can't find files on web, so the code is pasted below as well.

> Hope it helps, Klemen

>

> <http://www.sciencedirect.com/science/article/pii/S0098300403001043>

> www.utsa.edu/LRSG/Teaching/EES5053-06/project/cynthiaproject_remote.ppt

>

>

>

> ; Authors:

> ; Hongjie Xie, Nigel Hicks, G. Randy Keller, Haitao Huang, Vladik Kreinovich

> ;

> ; Title of the paper:

> ; An IDL/ENVI implementation of the FFT Based Algorithm for Automatic Image Registration

> ;

> ; IDL source code for shift, rotation and scaling

> ;

> ; -----
> ; Functions and procedures:

>

> FUNCTION Ritio, ARR1, ARR2

> ArrSize = SIZE(ARR1)

> Cols = ArrSize[1]

> Rows = ArrSize[2]

>

> R=COMPLEXARR(Cols,Rows)

> R1=COMPLEXARR(Cols,Rows)

> R2=FLTARR(Cols,Rows)

>

> R1=ARR1*(conj(ARR2))

> R2=(abs(ARR1))*(abs(ARR2))

>

> for j = 0, Cols - 1 do begin

> for i = 0, Rows - 1 do begin

> JUNK = CHECK_MATH()

> !EXCEPT=0

> R[j, i] = R1[j, i] / R2[j, i]

```

>   if FINITE(FLOAT(R[j, i])) EQ 0 then begin
>   endif
>   if FINITE(imaginary(R[j, i])) EQ 0 then begin
>   endif
>   endfor
> endfor
> RETURN, R
> END
function ritio, arr1, arr2
  return, ARR1*(conj(ARR2))/(abs(ARR1))*(abs(ARR2))
end

> FUNCTION LogPolar, RECT, LP
>   Pi = 3.14159365359
>   RArrSize = SIZE(RECT)
>   RCols = RArrSize[1]
>   RRows = RArrSize[2]
>   LP=temporary(FLTARR(RCols, RRows,/Nozero))
>
>   dTheta= 1.0 * pi / RRows      ; the angle
>   b = 10 ^ (alog10(RCols) / RCols)
>   for i=0.0, RRows-1.0 do begin
>     Theta=i * dTheta
>     for j=0.0, RCols-1.0 do begin
>       r = b ^ j - 1
>       x=r * cos(Theta) + RCols / 2.0
>       y=r * sin(Theta) + RRows / 2.0
>       x0=floor(x)
>       y0=floor(y)
>       x1=x0+1
>       y1=y0+1
>       if (x0 LE RCols-1) and (y0 LE RRows-1) and (x0 GT 1) and (y0 GT 1) THEN
V00=RECT[x0,y0] else V00=0.0
>       if (x0 LE RCols-1) and (y1 LE RRows-1) and (x0 GT 1) and (y1 GT 1) THEN
V01=RECT[x0,y1] else V01=0.0
>       if (x1 LE RCols-1) and (y0 LE RRows-1) and (x1 GT 1) and (y0 GT 1) THEN
V10=RECT[x1,y0] else V10=0.0
>       if (x1 LE RCols-1) and (y1 LE RRows-1) and (x1 GT 1) and (y1 GT 1) THEN
V11=RECT[x1,y1] else V11=0.0
>       V=V00*(x1-x)*(y1-y)+V01*(x1-x)*(y-y0)+V10*(x-x0)*(y1-y)+V11* (y-y0)*(x-x0)
>       LP[j,i]=temporary(V)
>     endfor
>   endfor
>   RETURN, LP
> END
FUNCTION LogPolar, RECT, LP
RArrSize = SIZE(RECT)
RCols = RArrSize[1]

```

```

RRows = RArrSize[2]
Theta=rebin(findgen(1,RRows)*(!const.pi/RRows),[RCols,RRows] ,/sample)
r =rebin(RCols^(findgen(RCols)/RCols),[RCols,RRows],/sample)
x=r * cos(Theta) + RCols / 2.0
y=r * sin(Theta) + RRows / 2.0
x0=floor(x)
y0=floor(y)
x1=x0+1
y1=y0+1
V00=RECT[0>x0<(RCols-1),0>y0<(RRows-1)]
V00[ where( ~( (x0 LE RCols-1) and (y0 LE RRows-1) and (x0 GT 1) $
and (y0 GT 1) ),/null ) ]=0.
V01=RECT[0>x0<(RCols-1),0>y1<(RRows-1)]
V01[ where( ~( (x0 LE RCols-1) and (y1 LE RRows-1) and (x0 GT 1) $
and (y1 GT 1) ),/null ) ]=0.
V10=RECT[0>x1<(RCols-1),0>y0<(RRows-1)]
V00[ where( ~( (x1 LE RCols-1) and (y0 LE RRows-1) and (x1 GT 1) $
and (y0 GT 1) ),/null ) ]=0.
V11=RECT[0>x1<(RCols-1),0>y1<(RRows-1)]
V11[ where( ~( (x1 LE RCols-1) and (y1 LE RRows-1) and (x1 GT 1) $
and (y1 GT 1) ),/null ) ]=0.
return, V00*(x1-x)*(y1-y)+V01*(x1-x)*(y-y0)+V10*(x-x0)*(y1-y) $
+V11*(y-y0)*(x-x0)
end

> PRO HighPass, ARR
> Pi = 3.14159365359
> ArrSize = SIZE(ARR)
> Cols = ArrSize[1]
> Rows = ArrSize[2]
> hpMatrix = temporary(FLTARR((Cols+1)*(Rows+1)))
> for i=0, Rows do begin
>   for j=0, Cols do begin
>     x=cos(Pi*((i-(Cols/2.0))*(1.0/Cols)))*cos(Pi*((j-(Rows/2.0)) *(1.0/Rows)))
>     hpMatrix[i*(Cols+1)+j]=(1.0-x)*(2.0-x)
>   endfor
> endfor
> for j=0, Cols-1 do begin
>   for i=0, Rows-1 do begin
>     ARR[j,i]=ARR[j,i]*hpMatrix[i*(Cols+1)+j]
>   endfor
> endfor
> END
PRO HighPass, ARR
ArrSize = SIZE(ARR)
Cols = ArrSize[1]
Rows = ArrSize[2]
x= rebin(cos(!const.pi*(findgen(1,Rows+1)/Cols-.5)),[Cols+1,Row s+1])* $

```

```

    rebin(cos(!const.pi*(findgen(Cols+1,1)/Rows-.5)),[Cols+1,Rows+1])
hpMatrix=(1.0-x)*(2.0-x)
ARR*=hpMatrix[0:-2,0:-2]
END

```

```

> PRO CombineFFT, ARR1, ARR2, FFTARR1, FFTARR2
>   ArrSize = SIZE(ARR1)
>   Cols = ArrSize[1]
>   Rows = ArrSize[2]
>   combine = FLTARR(Cols*2,Rows)
>   for j = 0, Cols - 1 do begin
>     for i = 0, Rows - 1 do begin
>       combine[j*2,i] = ARR1[j,i]
>       combine[j*2+1,i] = ARR1[j,i]
,,,,,,,,,,,,,; should be : combine[j*2+1,i] = ARR2[j,i]
>     endfor
>   endfor
>   fft_combine = fft(combine, -1)
>   for j = 0, Cols - 1 do begin
>     for i = 0, Rows - 1 do begin
>       FFTARR1[j,i] = fft_combine[j*2,i]
>       FFTARR2[j,i] = fft_combine[j*2+1,i]
>     endfor
>   endfor
> END

```

```

PRO CombineFFT, ARR1, ARR2, FFTARR1, FFTARR2
ArrSize = SIZE(ARR1)
Cols = ArrSize[1]
Rows = ArrSize[2]
fft_combine=fft(reform([reform(arr1,[1,Cols,Rows]), $
                        reform(arr2,[1,Cols,Rows])],[2*Cols,Rows]),-1)
fftarr1=fft_combine[0:-2:2,*]
fftarr2=fft_combine[1:-1:2,*]
END

```

```

> FUNCTION LoadImage, FileName, COLS, ROWS
>   ARR = BYTARR(COLS, ROWS)
>   get_lun, data_lun1
>   openr, data_lun1, filepath(FileName,root_dir=['D:\_code\Scratch\shift'])
>   readu, data_lun1, ARR
>   close, data_lun1
>   free_lun, data_lun1
>   RETURN, ARR
> END
>
> FUNCTION Normalize, ARR1
>   ArrSize = SIZE(ARR1)
>   Cols = ArrSize[1]

```

```

> Rows = ArrSize[2]
> ARR2 = temporary(FLTARR(COLS, ROWS,/Nozero))
> ARR2 = temporary(ARR1/255.0)
> RETURN, ARR2
> END
FUNCTION Normalize, ARR1
    return, ARR1/255.
END

> PRO ShiftArr, ARR
> ArrSize = SIZE(ARR)
> Cols = ArrSize[1]
> Rows = ArrSize[2]
> for j = 0, Cols - 1 do begin
>     for i = 0, Rows - 1 do begin
>         if ((i+j)mod 2) EQ 1 then begin
>             Arr[j,i]=temporary(Arr[j,i]*(-1))
>         endif
>     endfor
> endfor
> END
PRO ShiftArr, ARR
    ArrSize = SIZE(ARR)
    Cols = ArrSize[1]
    Rows = ArrSize[2]
Arr[where(rebin(lindgen(Cols,1),[Cols,Rows],/sample)+ $
    rebin(lindgen(1,Rows),[Cols,Rows],/sample) mod 2 eq 1)]=-1
END
>
>
> ;=====MAIN=====
=====
>
>
>
> PRO srs_idl, l1, l2, results=results;, Ffft_tm, Ffft_rad, Ffft_radc, absF_tm, absF_rad, $
> ;      R, Rc, lRc, maxn, lX, lY, lI, y, x, polar_coord1, $
> ;      n, m, base, plm1, plm2, i, j, V00, V01, V10, V11, $
> ;      X0, X1, y0, y1, s, R1, R2, R3, R1_Real, R1_Img
>
> n=512 ;column
> m=512 ; row
>
> ;print, 'Loading images...'
> ;l1 = LoadImage('t400.img', n, m)
> ;l2 = LoadImage('Tr19s1.3.img', n, m)
> ;
> l1 = Normalize(l1)

```

```

> Im2 = Normalize(I2)
>
> ShiftArr, Im1
> ShiftArr, Im2
>
> print, 'Doing FFT on two images...'
>
> fft_im1=FFT(im1,-1)
> fft_im2=FFT(im2,-1)
>
> print, 'Getting absolute value...'
>
> absF_im1=abs(fft_im1)
> absF_im2=abs(fft_im2)
>
> HighPass, absF_im1
> HighPass, absF_im2
>
> print, 'Transformming log_polar coordinates...'
> plm1 = LogPolar(absF_im1)
> plm2 = LogPolar(absF_im2)
>
> print, 'Doing FFT on polar images...'
>
> fft_polar1=fft(plm1,-1)
> fft_polar2=fft(plm2,-1)
>
> R = Ritio(fft_polar1, fft_polar2)
> inv_R = FFT(R, 1)
>
> abs_IR = abs(inv_R)
> Rim = imaginary(inv_R)
>
> maxn = MAX(abs_IR, position)
> ArrSize = SIZE(abs_IR)
> cols = ArrSize[1]
> rows = ArrSize[2]
> num = ArrSize[4]
>
> print, "
> print, "
> print, 'max absolute value position=', position
> print, 'total number of elements', num
> print, 'max absolute value =', maxn
>
> maxi=max(Rim, iii)
> print, 'max imaginary =', maxi
> print, 'max imaginary position=', iii

```

```

>
> b = 10 ^ (alog10(Cols) / Cols)
> scale = b^(position MOD rows)
> angle = 180.0 * (position / rows) / cols
>
> ang=angle
> if (ang eq 0.0) then begin
>
>   angle=angle
>
>   endif else if (ang ge 90.0) then begin
>     ang=angle+180.0
>     angle=180.0-angle
>
>   endif else if (ang lt 90.0) then begin
>     ang=angle
>     angle=-angle
>
>   endif
>
>
>
> print, 'computed scale =', scale
> print, 'computed angle =', angle
> ;print, 'input angle =', theta
>
> Im3=ROT(I2, -ang, 1.0/scale, /cubic);, MISSING = 0.0);, /PIVOT)
>
> Ffft_im1=fft(I1,-1)
> Ffft_im2=fft(Im3,-1)
>
> R1=(Ffft_im1*temporary(conj(Ffft_im2)))
> R2=(temporary(abs(Ffft_im1))*temporary(abs(Ffft_im2)))
> R=R1/R2
>
> IR=fft(R, 1)
> print, 'this is IR'
>
> maxn=MAX(IR,I)
>
> print, 'i'
> print, I
>
> IX=I MOD n
> IY=I / n
>
> X=IX
> Y=IY

```

```

>
> if ((X gt 0.5*n) and ( Y gt 0.5*m)) then begin
>   X=X-n
>   Y=Y-m
>   IX=X
>   IY=Y
>
>   endif else if (X gt 0.5*n) then begin
>     X=X-n
>     IX=X
>     IY=IY
>   endif else if ( Y gt 0.5*m) then begin
>     Y=Y-m
>     IX=IX
>     IY=Y
>   endif else begin
>     IX=IX
>     IY=IY
>   endelse
>
>
>
> print,'this is max'
> print, maxn
> print,'position'
> print, IX, IY
> results = [IX, IY, angle, scale]
>
> END
>
>
>
>
> ; Authors:
> ; Hongjie Xie, Nigel Hicks, G. Randy Keller, Haitao Huang, Vladik Kreinovich
> ;
> ; Title of the paper:
> ; An IDL/ENVI implementation of the FFT Based Algorithm for Automatic Image Registration
>
>
> ; IDL source code for shift only
> ; -----
>
>
> pro shift_idl, im1, im2, results=results;Ffft_tm, Ffft_rad, Ffft_radc, absF_tm, absF_rad, $
> ;      R, Rc, IRc, maxn, IX, IY, I
>
> n=512 ; pixel columns
> m=512 ; pixel lines

```

```

>
> ;im1=bytarr(n,m,/Nozero)
> ;get_lun, data_lun1
> ;openr, data_lun1, filepath('t350380ori.img',root_dir=['D:\_code\Scratch\shift' ]) ;open original
image for reading
> ;readu, data_lun1, im1
> ;close, data_lun1
> ;free_lun, data_lun1
> Ffft_im1=fft(im1,-1) ; forward FFT
>
> ;im2=bytarr(n,m,/Nozero)
> ;get_lun, data_lun2
> ;openr, data_lun2, filepath('t350380shf.img', root_dir=['D:\_code\Scratch\shift']) ;open shift
image for reading
> ;readu, data_lun2, im2
> ;close, data_lun2
> ;free_lun, data_lun2
> Ffft_im2=fft(im2,-1) ; forward FFT
>
> R1=(Ffft_im1*temporary(conj(Ffft_im2)))
> R2=(temporary(abs(Ffft_im1))*temporary(abs(Ffft_im2)))
> R=R1/R2 ; calculate the ratio
>
> IR=fft(R, 1) ; inverse FFT
>
> maxn=MAX(IR,I) ; get the position of maximum IR
> print, 'i'
> print, I
>
> IX=I MOD n ; get the shift in columns
> IY=I / n ; get the shift in lines
>
> X=IX
> Y=IY
>
> if ((X gt 0.5*n) and ( Y gt 0.5*m)) then begin
>   X=X-n
>   Y=Y-m
>   IX=X
>   IY=Y
>
> endif else if (X gt 0.5*n) then begin
>   X=X-n
>   IX=X
>   IY=IY
> endif else if ( Y gt 0.5*m) then begin
>   Y=Y-m
>   IX=IX

```

```
>   IY=Y
>   endif else begin
>     IX=IX
>     IY=IY
>   endelse
>
>   print,'this is max'
>   print, maxn
>   print,'position'
>   print, IX, IY
>   results = [IX, IY]
>
>   end
>
```
