

Hi,

I've been trying for a very long time now to implement a principle axes transformation for the coregistration of MR image volumes. My rather simplistic understanding of how this should work is:

Find centre of mass
Calculate inertia matrix of image in centre of mass coordinate system
Find eigenvalues and eigenvectors of inertia matrix
Matrix multiply centre of mass coordinates by eigenvectors then
transform back to cartesian coordinates

Voila image is rotated onto it's principle axes!

I should mention that I have no natural feel for maths and my 'play with it till it works' approach to life has proved to be a dismal failure in this project. I've tried many variations of code in the last months to try to get the principle axes transform to work for 3D matrices. Including a hack of Martin Downing's stunning `transform_volume`. I've included the inertia matrix code, and principle axes code (minus all the widget state variable stuff). These are loop versions as it may be easier for someone to spot any obvious screw ups on my part, and the matrix multiplication is (I think) quaternion, as playing with that seemed to give something close to registration.

If anyone has any hints or observations (give up is acceptable, and very close :) I'd be more than happy to hear them.

Thanks

Marc O'Brien

```
function Inertia3D, imagearray, com
```

```
; Calculate inertia matrix for image
sarr = size(imagearray)
x = sarr[1]
y = sarr[2]
z = sarr[3]
npix = x * y * z
```

```
inertia = dblarr(3,3)
```

```
for i=-com[0], (x-com[0])-1 do begin
```

```

for j=-com[1], (y-com[1])-1 do begin
for k=-com[2], (z-com[2])-1 do begin
  pixelval = getpix_3dcom(i,j,k,com,imagearray)

```

```

; Moment of inertia = mass * distance ^ 2

```

```

; xx

```

```

temp = pixelval * (j^2 + k^2)

```

```

inertia[0,0] = inertia[0,0] + temp

```

```

; yy

```

```

temp = pixelval * (k^2 + i^2)

```

```

inertia[1,1] = inertia[1,1] + temp

```

```

; zz

```

```

temp = pixelval * (i^2 + j^2)

```

```

inertia[2,2] = inertia[2,2] + temp

```

```

; Products of inertia

```

```

; xy

```

```

temp = (pixelval * i * j)

```

```

inertia[0,1] = inertia[0,1] + temp

```

```

; xz

```

```

temp = (pixelval * i * k)

```

```

inertia[0,2] = inertia[0,2] + temp

```

```

; yz

```

```

temp = (pixelval * j * k)

```

```

inertia[1,2] = inertia[1,2] + temp

```

```

endfor

```

```

endfor

```

```

endfor

```

```

; Get mean

```

```

inertia[0,0] = inertia[0,0] / npix

```

```

inertia[1,1] = inertia[1,1] / npix

```

```

inertia[2,2] = inertia[2,2] / npix

```

```

inertia[0,1] = (inertia[0,1]) / npix

```

```

inertia[0,2] = (inertia[0,2]) / npix

```

```

inertia[2,1] = (inertia[2,1]) / npix

```

```

; ZX

```

```

inertia[2,0] = inertia[0,2]

```

```

; YX

```

```

inertia[1,0] = inertia[0,1]

```

```

; ZY

```

```
inertia[1,2] = inertia[2,1]
```

```
return, inertia  
end
```

```
pro OnPax3D, event
```

```
<snip>
```

```
; Volumes are vol and nc although for testing both volumes have  
; been the same with one rotated
```

```
; Get volume data
```

```
ivol = volarray[*,*,*]
```

```
inc = nccom[*,*,*]
```

```
sizevol = size(ivol)
```

```
sizenc = size(inc)
```

```
; Get centers of mass
```

```
volcom=dblarr(3)
```

```
nccom=dblarr(3)
```

```
nccom=ROUND(COM_n(inc, /DOUBLE))-1
```

```
volcom=ROUND(COM_n(ivol,/DOUBLE))-1
```

```
; Calculate inertia matrix for vol image
```

```
I1 = Inertia3D(ivol,volcom)
```

```
; Calculate inertia matrix for nc image
```

```
I2 = Inertia3D(inc,nccom)
```

```
; Get eigencolumns for volume image
```

```
S1 = I1
```

```
TRIRED, S1, D1, E, /DOUBLE
```

```
TRIQL, D1, E, S1, /DOUBLE
```

```
; Get eigencolumns for non contrast volume
```

```
S2 = I2
```

```
TRIRED, S2, D2, F, /DOUBLE
```

```
TRIQL, D2, F, S2, /DOUBLE
```

```
; Create the rotation matrices
```

```
;R1 = S1 ## TRANSPOSE(S2)
```

```
;R2 = S2 ## TRANSPOSE(S1)
```

```
R1 = TRANSPOSE(S1)
```

```
R2 = TRANSPOSE(S2)
```

```
; Create arrays to store transformed images
```

```
newvol = dblarr(sizevol[1],sizevol[2],sizevol[3])
```

```
newnc = dblarr(sizenc[1],sizenc[2],sizenc[3])
```

```

; Get array dimension sizes
sxvol = sizevol[1]
syvol = sizevol[2]
szvol = sizevol[3]
sxnc = sizenc[1]
sync = sizenc[2]
sznc = sizenc[3]

quartmat1 = dblarr(4,4)
quartmat1[3,*] = 0
quartmat1[* ,3] = 0
quartmat1[3,3] = 1
quartmat1[0:2,0:2] = R1
quartmat2 = dblarr(4,4)
quartmat2[3,*] = 0
quartmat2[* ,3] = 0
quartmat2[3,3] = 1
quartmat2[0:2,0:2] = R2

print, quartmat1
print
print, quartmat2

quartvec = TRANSPOSE([1,1,1,1])
newquartvec = TRANSPOSE([1,1,1,1])

; Back to the loops
for z=0, szvol-1 do begin
  for y=0, syvol-1 do begin
    for x=0, sxvol-1 do begin
      curpix = [x,y,z]

      ; Convert to centre of mass coordinates
      curpix = curpix - volcom
      quartvec[0:2] = curpix
      newquartvec = quartmat1 ## quartvec
      ; Convert back to cartesian
      newquartvec[0:2] = newquartvec[0:2] + volcom
      newquartvec = ABS(FIX(newquartvec))
      ; Check for outliers
      if newquartvec[0] GE sxvol then newquartvec[0] = sxvol -1
      if newquartvec[1] GE syvol then newquartvec[1] = syvol -1
      if newquartvec[2] GE szvol then newquartvec[2] = szvol -1
      ; Copy pixel location
      newvol[x,y,z] = ivol[newquartvec[0],newquartvec[1],newquartvec[2]]

      ; Repeat for second image

```

```
curpix = [x,y,z]
curpix = curpix - nccom
quartvec[0:2] = curpix
newquartvec = quartmat2 ## quartvec
newquartvec[0:2] = newquartvec[0:2] + nccom
newquartvec = ABS(FIX(newquartvec))
if newquartvec[0] GE sxnc then newquartvec[0] = sxnc -1
if newquartvec[1] GE sync then newquartvec[1] = sync -1
if newquartvec[2] GE sznc then newquartvec[2] = sznc -1
newnc[x,y,z] = inc[newquartvec[0],newquartvec[1],newquartvec[2]]
endfor ; x
endfor ; y
endfor ; z
<snip>
end
```

--
Tel: 0208 725 2857
Fax: 0208 725 2992

St George's Hospital Medical School
Department of Biochemistry and Immunology
Cranmer Terrace
Tooting
London SW17 0RE
