Subject: Re: x-y offsets
Posted by Jeremy Bailin on Thu, 20 May 2010 02:56:53 GMT
View Forum Message <> Reply to Message

On May 19, 9:38 pm, Jeremy Bailin <astroco...@gmail.com> wrote:
> On May 19, 7:45 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>
>
>
>
>> On May 19, 2:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>>> On May 18, 8:29 pm, Gray <grayliketheco...@gmail.com> wrote:
>
>>>> Hi all,
>
>>>> This is a variation on the 2D matching problem that I'm having trouble
>>>> algorithm-ing (to coin an incredibly awkward word).
>
>>>> I have two sets of XY coordinates of unequal length (i.e., x1/y1/n1,
>>>> x2/y2/n2, n1 ne n2). I want to find offsets in both X and Y that
>>>> match the two sets as closely as possible (there will obviously be
>>>> some unmatched coordinates in the larger set). I'm just looking for
>>>> constant offsets, so basically (for n1 < n2) x1 + Cx -> x2, y1 + Cy ->
>>>> y2, with some elements of x2 and y2 being unmatched. How do I go
>>>> about doing this? I don't think I can use JD's MATCH_2D because I
>>>> don't know a priori what my matching radius is.
>
>>>> Any suggestions? Thanks, as always!
>
>>>> --Gray
>
>>> I would be tempted to create a 2D histogram based on each set and then
>>> cross-correlate them.
>
>>> -Jeremy.
>
>> How do you turn the cross-correlation into offsets? And, how do you
>> intelligently choose a binsize for the histogram?
>
> The first question is the easier one. ;-)
>
> IDL> d = dist(5,5)
> IDL> a = fltarr(25,25)
> IDL> b = fltarr(25,25)
> IDL> a[4,7] = d
> IDL> b[0,0] = d

> IDL> xcor = fft(/inverse, fft(a)*fft(b,/inverse))
> IDL> maxcor = max(abs(xcor),loc)
> IDL> print, array_indices(a,loc)
> 4 7
>
> Now, it's easy here because I know that there's one perfect matching
> location - it may be more ambiguous in a real situation (in which case
> you'll probably to assess the magnitude of all of the peaks within
> xcor to see if there are multiple plausible solutions). Also note that
> the answer wraps around - i.e. you should treat a value of 24 here as
> -1.
>
> As for the binsize, it depends on your application. Ideally you would
> make the bins as small as the precision you expect to be able to
> achieve in determining the translational offset given your data (or
> even better, a factor of two smaller) - but if that means that your 2D
> histograms have one million bins in each direction then that won't
> work. ;-) So in that case, I would go for a two-step process: in step
> 1, use the cross-correlation of the entire image using a coarse grid
> to get in the right ballpark. Then, if you think you should be good to
> within a length L, do a finer resolution cross-correlation just using
> a box of length L around each point (you might be able to ram the
> boxes all up against each other in a big image so you can do the cross-
> correlation of them all at once - never tried it).
>
> -Jeremy.

Here's a quick implementation. As you can see, it gets to within half
of the
input scatter. I'm sure you could do quite a bit better by, instead of
just using the peak location of the cross-correlation, fit a 2D
Gaussian
to it to find the peak location to well within one bin width.

Actually, I really like Craig's algorithm. I would probably say that
in
my two-step suggestion above, use the cross-correlation as the coarse
step
and then Craig's suggestion as the fine step.

```
seed=43l
n1 = 100l
n2 = 500l

offset = [0.3, -0.15]
scatter = 0.02
subset = floor(randomu(seed,n1)*n2)
```

```
; generate random positions
x2 = randomu(seed,n2)
y2 = randomu(seed,n2)
; for a subset of them, offset them and add scatter
x1 = x2[subset] + offset[0] + scatter*randomu(seed,n1)
y1 = y2[subset] + offset[1] + scatter*randomu(seed,n1)

bin = 0.5*scatter
xrange = minmax([x1,x2])
xrange[0] -= bin & xrange[1] += bin
yrange = minmax([y1,y2])
yrange[0] -= bin & yrange[1] += bin

; create 2D histogram of each distribution - effectively an
; image that we can match
image1 = hist_2d(x1, y1, min1=xrange[0],max1=xrange[1],bin1=bin, $
  min2=yrange[0],max2=yrange[1],bin2=bin)
image2 = hist_2d(x2, y2, min1=xrange[0],max1=xrange[1],bin1=bin, $
  min2=yrange[0],max2=yrange[1],bin2=bin)
imagesize = size(image1,/dimen)

; calculate cross-correlation by FT convolution theorem
xcor = fft(/inverse, fft(image1)*fft(image2,/inverse))
; the peak in the cross-correlation is where they match best
maxcor = max(abs(xcor),loc)
maxindex = array_indices(image1,loc)
; the FT is periodic, so locations >1/2 of the image size should
; be considered as negative offsets from the edge of the image
for i=0,1 do if maxindex[i] gt imagesize[i]/2 then maxindex[i] -=
imagesize[i]

measuredoffset = maxindex * bin

print, 'Input offsets:',offset
print, 'Measured offsets:',measuredoffset

!p.multi=[0,2,1]
red=fsc_color('red')
plot, psym=3, xrange=xrange, yrange=yrange, x2, y2, title='Original', /
iso
oplot, psym=3, x1, y1, color=red
plot, psym=3, xrange=xrange, yrange=yrange, x2, y2, title='Matched', /
iso
oplot, psym=3, x1-measuredoffset[0], y1-measuredoffset[1], color=red
```

-Jeremy.