Subject: Re: x-y offsets
Posted by Gray on Tue, 18 May 2010 21:08:55 GMT
View Forum Message <> Reply to Message

On May 18, 4:29 pm, Gray <graylikethecо...@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

To clarify, there can be unmatched coordinates in both lists, but one
of them is guaranteed to have unmatched coords unless n1 = n2 (which
is possible in my scenario, but unlikely).

Subject: Re: x-y offsets
Posted by Jeremy Bailin on Wed, 19 May 2010 06:50:58 GMT
View Forum Message <> Reply to Message

On May 18, 8:29 pm, Gray <graylikethecо...@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.

Subject: Re: x-y offsets
Posted by Gray on Wed, 19 May 2010 19:45:13 GMT
View Forum Message <> Reply to Message

On May 19, 2:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> On May 18, 8:29 pm, Gray <graylikethec...@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?

Subject: Re: x-y offsets
Posted by Jeremy Bailin on Thu, 20 May 2010 01:38:08 GMT
View Forum Message <> Reply to Message

On May 19, 7:45 pm, Gray <graylikethec...@gmail.com> wrote:
> On May 19, 2:50 am, Jeremy Bailin <astroco...@gmail.com> wrote:
>
>
>
>
>
>
>> On May 18, 8:29 pm, Gray <graylikethec...@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.

Subject: Re: x-y offsets
Posted by Craig Markwardt on Thu, 20 May 2010 01:45:18 GMT
View Forum Message <> Reply to Message

On May 18, 4:29 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!

This seems like a fragile problem. It seems like the key thing is to try to select the matching pairs first, and then you can try to refine

the offset determination. If you match incorrectly, then it's possible for a spoiler pair to corrupt the offset refinement. If you had an a priori guess for the offset, that would help immensely.

As far as selecting pairs...

Compute the distances between all pairs of points. For your sample, that would be N1*N2 distance calculations. The result would be a list of N1*N2 distances. Presumably the "correct" offset would appear most frequently. One way to check this is make a histogram of the distances and look for a peak. Here, having an a priori guess is helpful to choose the bin size and histogram range. Otherwise you might have to try those iteratively.

Once you have a preferred offset distance, then you should be able to go through and match pairs reasonably efficiently. Also, you can enforce the constraint that a pair can only match once.

From that point, it should be easy to compute the mean offset, or if you have error bars, then the weighted mean offset. But you will probably have to check for outliers, which correspond with poorly matched pairs, and then iteratively recompute offset.

As I said, "fragile."

Craig

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.

Subject: Re: x-y offsets
Posted by Jeremy Bailin on Thu, 20 May 2010 04:59:26 GMT
View Forum Message <> Reply to Message

On May 20, 2:56 am, Jeremy Bailin <astroco...@gmail.com> wrote:
> 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.

Here's a more refined version that uses the Gaussian fit. Does
very well as long as the scatter doesn't reach the average interpoint
spacing of x2,y2 (0.045 in this example - at which point, the problem
is no longer very well-defined).

Tests:

scatter=0.005:
Input offsets: 0.349760 -0.151510
Measured offsets: 0.349591 -0.152183
Delta/scatter: 0.0338316 0.134644

scatter=0.01:
Input offsets: 0.349760 -0.151510
Measured offsets: 0.350363 -0.151164
Delta/scatter: 0.0603169 0.0345916

scatter=0.02:
Input offsets: 0.349760 -0.151510
Measured offsets: 0.341009 -0.149584
Delta/scatter: 0.437570 0.0963129

scatter=0.03:
% CURVEFIT: Failed to converge- CHISQ increasing without bound.
% CURVEFIT: Failed to converge- CHISQ increasing without bound.
Input offsets: 0.349760 -0.151510
Measured offsets: 0.342206 -0.185512
Delta/scatter: 0.251811 1.13339

scatter=0.05:

% CURVEFIT: Failed to converge- CHISQ increasing without bound.
Input offsets: 0.349760 -0.151510
Measured offsets: -0.101709 -0.336232
Delta/scatter: 9.02937 3.69445

Code:

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

offset = [0.34976, -0.15151]
scatter = 0.01
subset = floor(randomu(seed,n1)*n2)

x2 = randomu(seed,n2)
y2 = randomu(seed,n2)
x1 = x2[subset] + offset[0] + scatter*randomn(seed,n1)
y1 = y2[subset] + offset[1] + scatter*randomn(seed,n1)

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

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)

; cross-correlation via convolution theorem
xcor = fft(/inverse, fft(image1)*fft(image2,/inverse))
maxcor = max(abs(xcor),loc)
maxindex = array_indices(image1,loc)

; to fit gaussian, need to take periodicity into account
axcor = abs([[xcor,xcor,xcor],[xcor,xcor,xcor],[xcor,xcor,xcor]])
boxlen=7   ; size of box around peak which to fit
halfbox=boxlen/2
; put into middle image of 3x3
maxindex_periodic = maxindex + imagesize
aa = axcor[maxindex[0]-halfbox:maxindex[0]+halfbox, $
  maxindex[1]-halfbox:maxindex[1]+halfbox]
; 2D gaussian fit of boxed region
```

```
params = [0.,max(aa),1.,1.,halfbox,halfbox,0.]
yfit = gauss2dfit(aa, params)
; put back into original coordinates
refinedindex = params[4:5]-halfbox+maxindex
; deal with periodicity
for i=0,1 do if refinedindex[i] gt imagesize[i]/2 then $
  refinedindex[i] -= imagesize[i]

measuredoffset = refinedindex * bin

print, 'Input offsets:',offset
print, 'Measured offsets:',measuredoffset
print, 'Delta/scatter:',abs(measuredoffset-offset)/scatter

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

end
```

-Jeremy.

Subject: Re: x-y offsets
Posted by Gray on Thu, 20 May 2010 15:01:08 GMT
View Forum Message <> Reply to Message

Jeremy, Craig,

Thanks for the great suggestions! I've tried both methods, and here are my comments.

First, some more information about the problem - I wanted to ask it in as general a way as possible because I think it would be useful for others to have the answer for this question. However, what I really want is to find the offset, remove it, and match pairs to use as pins for poly-warping. As I said above, I couldn't figure out how to find a good match radius, and removing the offset should shrink the radius to something manageable. But, Craig's method could work too for finding my match radius, in which case I wouldn't need to find an offset at all (in all cases, I'm using MATCH_2D to match sources).

Neither method worked completely - I have a particular data set which broke both methods, so here's what we have (I can give out the actual

numbers if you like):

```
IDL> print, minmax(x1), minmax(y1)
     2.24139      128.413
     2.91512      122.837
IDL> print, minmax(x2), minmax(y2)
    0.949352      127.562
     7.84978      127.785
IDL> print, n_elements(x1), n_elements(x2)
          82          47
```

First I tried the cross-correlation/gauss2dfit method. I tweaked the algorithm very slightly - the change I'm most proud of (which was completely negligible) was to replace the periodicity for-loop with the line:
refinedindex -= imagesize * (refinedindex gt imagesize/2)
I did one pass only, using as my coarse binsize my desired match radius (this seemed intuitive, if someone can think of a better way to choose a binsize, let me know), and for a number of datasets it worked beautifully. However, about half the datasets failed to converge for the gaussian fit. In general this gave reasonable results (I think...), but for the dataset above the x offset was ~3e12, so I added a reasonableness check: if the offset x values are all greater or all less than the x-range (same for y), then use the simple max_index result. However, the points still didn't match up - there was still a systematic x offset that caused the matching to fail. Maybe doing another pass would fix that, but I haven't tried yet.

Then I tried Craig's distance-histogram (or "distogram", if you will) suggestion. First problem is that there's no guarantee that the "preferred" offset is actually the maximum of the full histogram - there's a predicable peak around half of the maximum distance between two points (for these sets, around 75). So, I have to pick a histogram range, but I don't know what it is likely to be a priori (which is the whole point of this exercise). However, let's assume that it's somewhere between a 0 pixel and 20 pixel shift, and then the distogram max should be the actual offset. With that in mind, there's a couple things I tried, both of which fail for the same reason. First was to just use MATCH_2D with that distance as the match radius; the other was to use reverse_indices to pick out the distance pairs that fell in that bin, then compute a mean offset. However, both methods run into the problem of multiple matches. MATCH_2D allows multiple mapping from x1/y1 onto x2/y2 (though it prevents the reverse) in the case of a large radius, which this is (~12 for this dataset), and the reverse_indices method does the same. Even if you try to discard multiple matches, there's no way to discriminate between them because there's no guarantee that the minimum distance match is the right one.

All in all, I think we're close, but no cigar yet. Thoughts?

Subject: Re: x-y offsets
Posted by Jeremy Bailin on Thu, 20 May 2010 19:20:01 GMT
View Forum Message <> Reply to Message

On May 20, 3:01 pm, Gray <graylikethec...@gmail.com> wrote:
> Jeremy, Craig,
>
> Thanks for the great suggestions! I've tried both methods, and here
> are my comments.
>
> First, some more information about the problem - I wanted to ask it in
> as general a way as possible because I think it would be useful for
> others to have the answer for this question. However, what I really
> want is to find the offset, remove it, and match pairs to use as pins
> for poly-warping. As I said above, I couldn't figure out how to find
> a good match radius, and removing the offset should shrink the radius
> to something manageable. But, Craig's method could work too for
> finding my match radius, in which case I wouldn't need to find an
> offset at all (in all cases, I'm using MATCH_2D to match sources).
>
> Neither method worked completely - I have a particular data set which
> broke both methods, so here's what we have (I can give out the actual
> numbers if you like):
>
> IDL> print, minmax(x1), minmax(y1)
>   2.24139   128.413
>   2.91512   122.837
> IDL> print, minmax(x2), minmax(y2)
>   0.949352   127.562
>   7.84978   127.785
> IDL> print, n_elements(x1), n_elements(x2)
>   82   47
>
> First I tried the cross-correlation/gauss2dfit method. I tweaked the
> algorithm very slightly - the change I'm most proud of (which was
> completely negligible) was to replace the periodicity for-loop with
> the line:
> refinedindex -= imagesize * (refinedindex gt imagesize/2)
> I did one pass only, using as my coarse binsize my desired match
> radius (this seemed intuitive, if someone can think of a better way to
> choose a binsize, let me know), and for a number of datasets it worked
> beautifully. However, about half the datasets failed to converge for
> the gaussian fit. In general this gave reasonable results (I
> think...), but for the dataset above the x offset was ~3e12, so I

> added a reasonableness check: if the offset x values are all greater
> or all less than the x-range (same for y), then use the simple
> max_index result. However, the points still didn't match up - there
> was still a systematic x offset that caused the matching to fail.
> Maybe doing another pass would fix that, but I haven't tried yet.

Here's one idea: when the sanity check for the gaussian fit fails,
iterate with a coarser bin size. If you take a look at the abs(xcor)
image for a bunch of different bin sizes, what you'll see is that as
the bin size goes down, the values get noisier, and so the chances
that
there exists an individual unrelated pixel that's higher than the
maximum of the real peak go up. If that happens, the gaussian fit
(which only looks at a 7x7 box around the "peak") will just fit to
noise and is likely to fail miserably. As you go to coarser bin sizes,
the chances of that go down but so does the precision with which you
can determine the offset.

> Then I tried Craig's distance-histogram (or "distogram", if you will)
> suggestion. First problem is that there's no guarantee that the
> "preferred" offset is actually the maximum of the full histogram -
> there's a predicable peak around half of the maximum distance between
> two points (for these sets, around 75). So, I have to pick a
> histogram range, but I don't know what it is likely to be a priori
> (which is the whole point of this exercise). However, let's assume
> that it's somewhere between a 0 pixel and 20 pixel shift, and then the
> distogram max should be the actual offset. With that in mind, there's
> a couple things I tried, both of which fail for the same reason.
> First was to just use MATCH_2D with that distance as the match radius;
> the other was to use reverse_indices to pick out the distance pairs
> that fell in that bin, then compute a mean offset. However, both
> methods run into the problem of multiple matches. MATCH_2D allows
> multiple mapping from x1/y1 onto x2/y2 (though it prevents the
> reverse) in the case of a large radius, which this is (~12 for this
> dataset), and the reverse_indices method does the same. Even if you
> try to discard multiple matches, there's no way to discriminate
> between them because there's no guarantee that the minimum distance
> match is the right one.

I think once you have a set of plausible distance pairs, it becomes
a minimization problem. How about if, once you have a set of
possible pairs from the "distogram" reverse_indices, you construct
a function that calculates the total distance squared between
all of the pairs and use something like POWELL to minimize it?

-Jeremy.

Subject: Re: x-y offsets
Posted by Jeremy Bailin on Thu, 20 May 2010 20:11:17 GMT
View Forum Message <> Reply to Message

> I think once you have a set of plausible distance pairs, it becomes
> a minimization problem. How about if, once you have a set of
> possible pairs from the "distogram" reverse_indices, you construct
> a function that calculates the total distance squared between
> all of the pairs and use something like POWELL to minimize it?

To be more specific, I would go with calculating the mean offset
using the reverse_indices to get you close, then use
MATCHALL_2D using the offset coordinates to get neighbours of
each x1,y1 pair. Then you can minimize the following function
for xoff,yoff:

Sum_i (min((x1_i-x2+xoff)^2 + (y1_i-y2+yoff)^2))

where i runs over x1,y1 and the minimization is over all x2,y2
points that are neighbours of x1_i,y1_i.

-Jeremy.