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 <graylikethecо...@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.