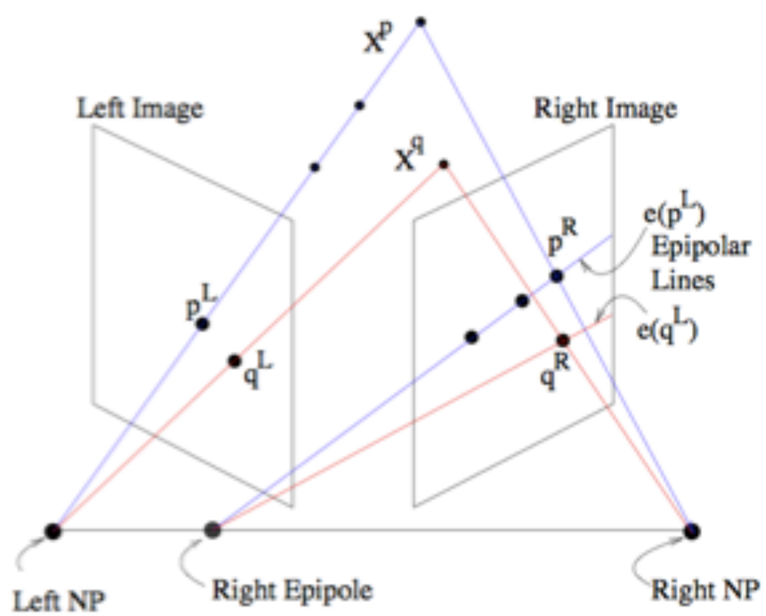
Projection, stereo and panoramic images

X^p and X^q are the physical location of objects.

NP are the nadir points of the cameras.



http://www.cs.toronto.edu/~jepson/csc420/notes/epiPolarGeom.pdf
(note, the image is posted on an educational site and copied here without following up on permissions. Any further use of the image should follow up on the origins and permissions.)

The projection of the left nadir, NP is seen as a the right epipole with w.r.t. the right image. The epipoles may or may not be within the border of the images.

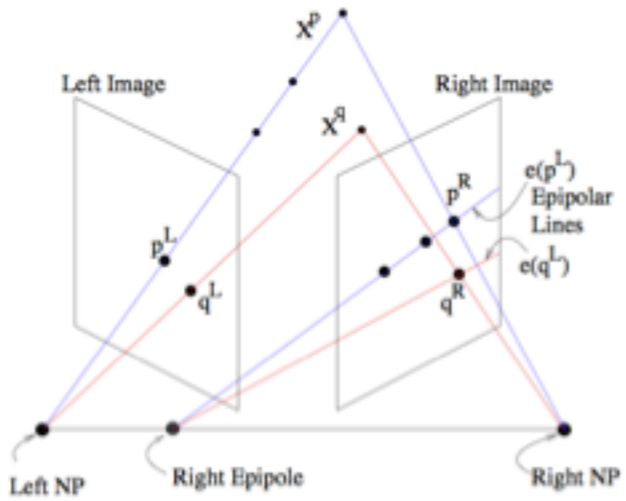
The projection of X^p to left nadir, NP is seen as an epipole line in the right image. If more than one epipole line is present in the right image, they converge at the right epipole (which might not be within the boundaries of the image).

Once the points in the left image, X_L are matched with points in the right image, X_R, if there are at least 7 points, one can determine the "bifocal tensor",
a.k.a. "fundamental matrix, relating the points in the 2 images using a 3x3 matrix of rank 2.

$(X_L)^T * F * X_R = 0$ for any pair or points in the images.

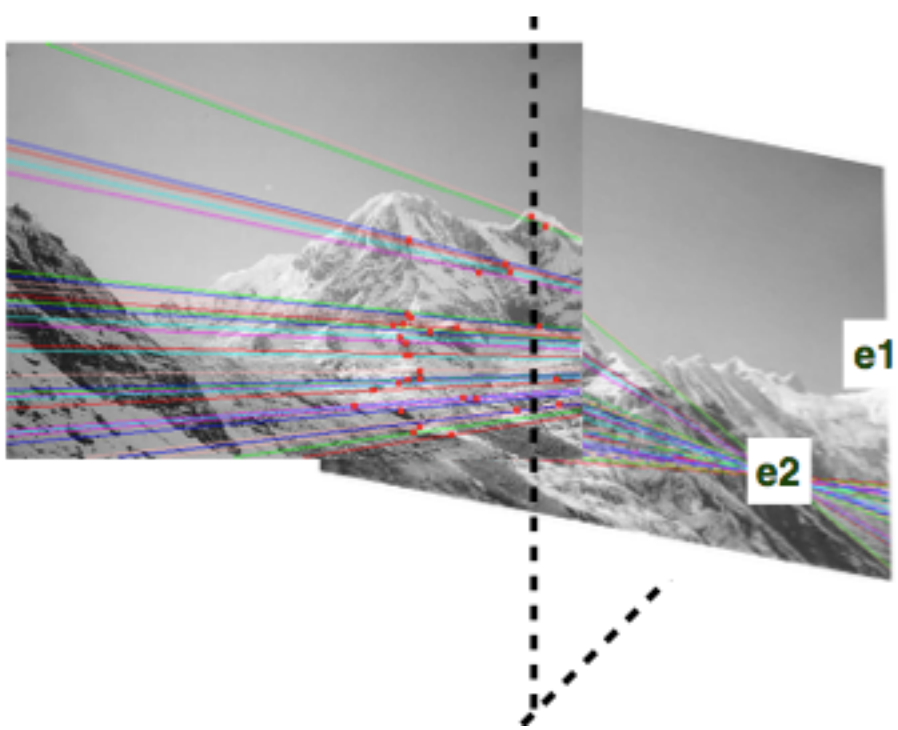
Note: the "Essential matrix" is a matrix used if the camera details are known. The "bifocal tensor", a.k.a. "fundamental matrix" does not need camera details.

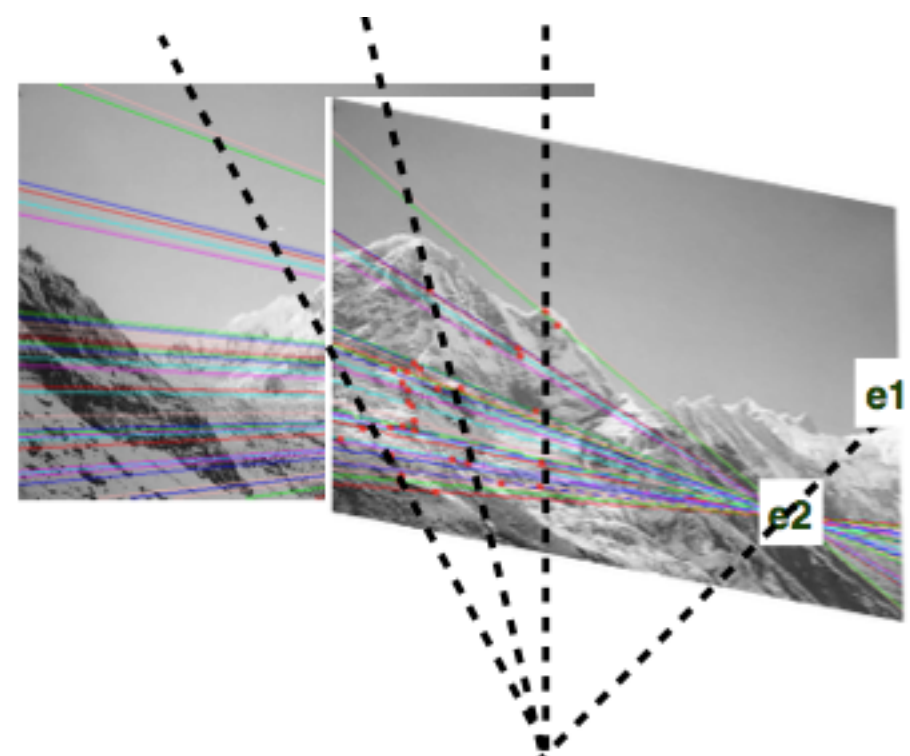
Projection, stereo and panoramic images



http://www.cs.toronto.edu/~jepson/csc420/notes/epiPolarGeom.pdf
(note, the image is posted on an educational site and copied here without following up on permissions. Any further use of the image should follow up on the origins and permissions.)

panoramic images here are from
Brown & Lowe 2003

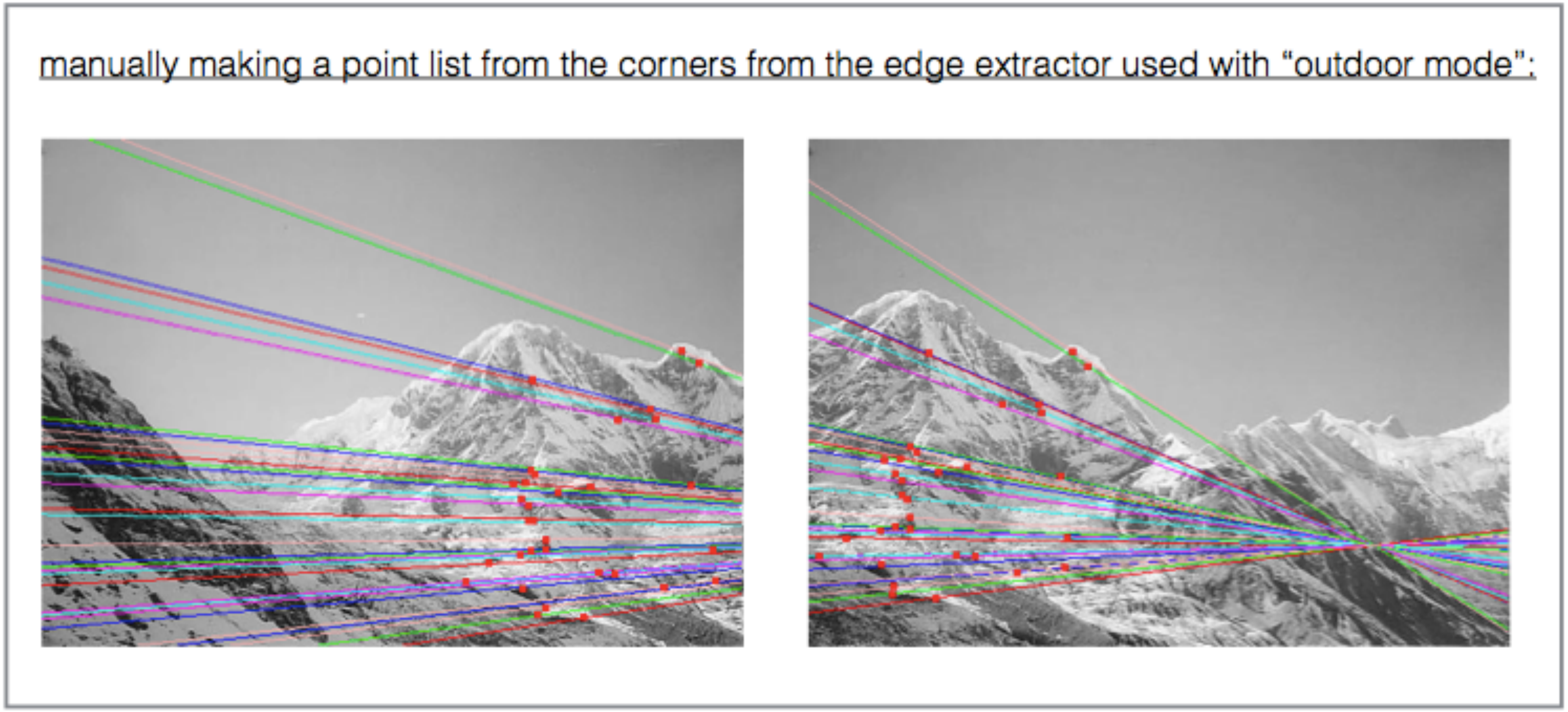




same camera objectives (nadirs), but
different orientation for the 2 images
(that is, rotated around the same nadir)

Point Correspondence

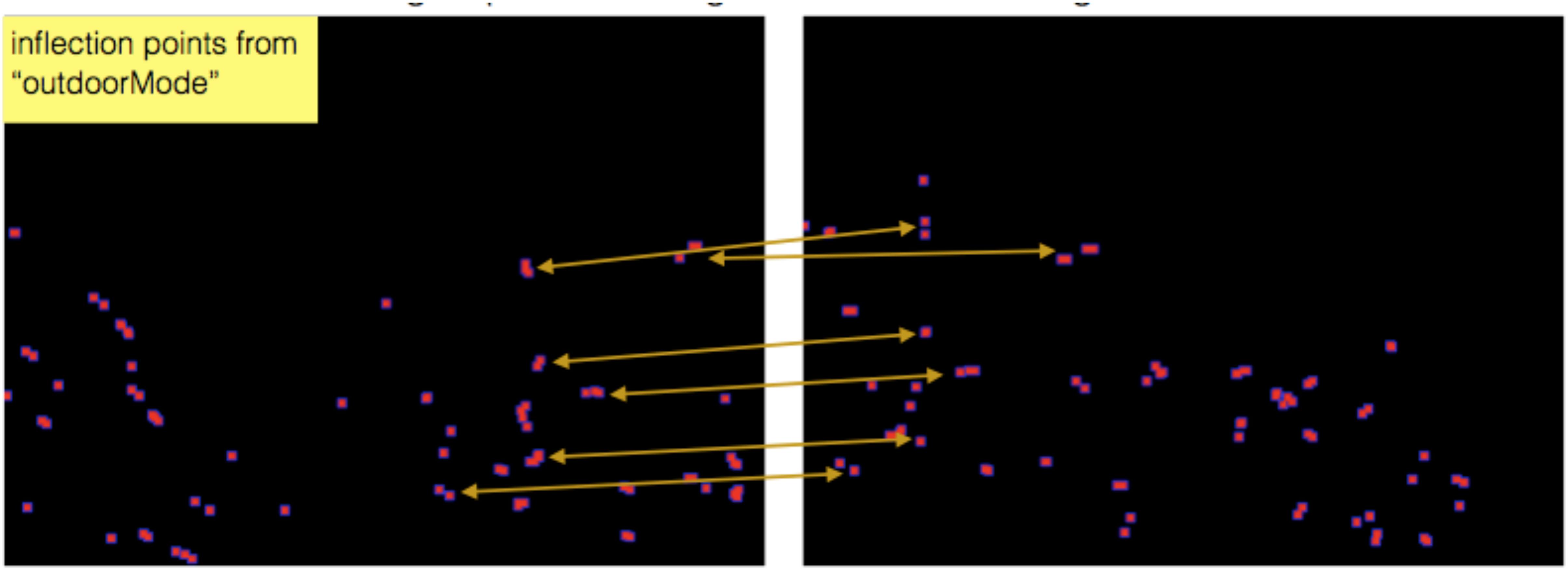
Need to create list of matched points between the images in order to solve for the epipolar projection



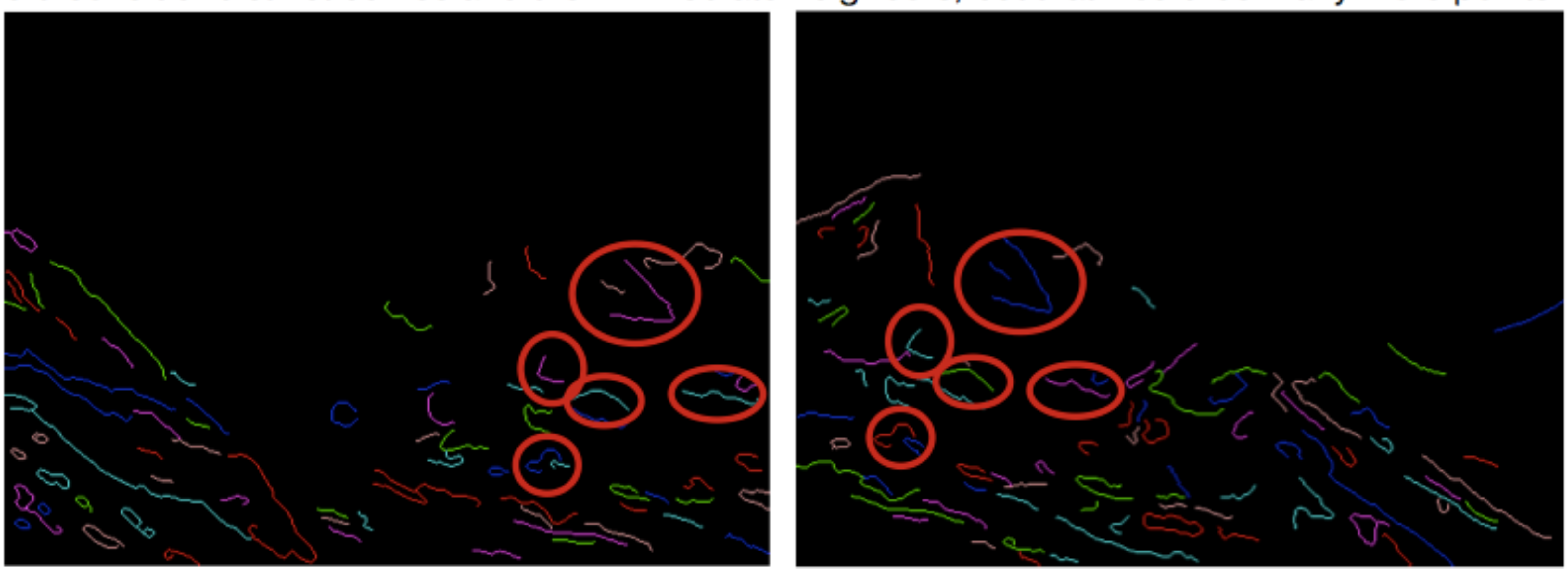
stereo projection fit to 32 points already known to match shows what the epipolar projections should be when the corner find + corner match + stereo projection solve are correctly automated.

```
nMatched=32
avgDist=0.281
stDev=0.508
```

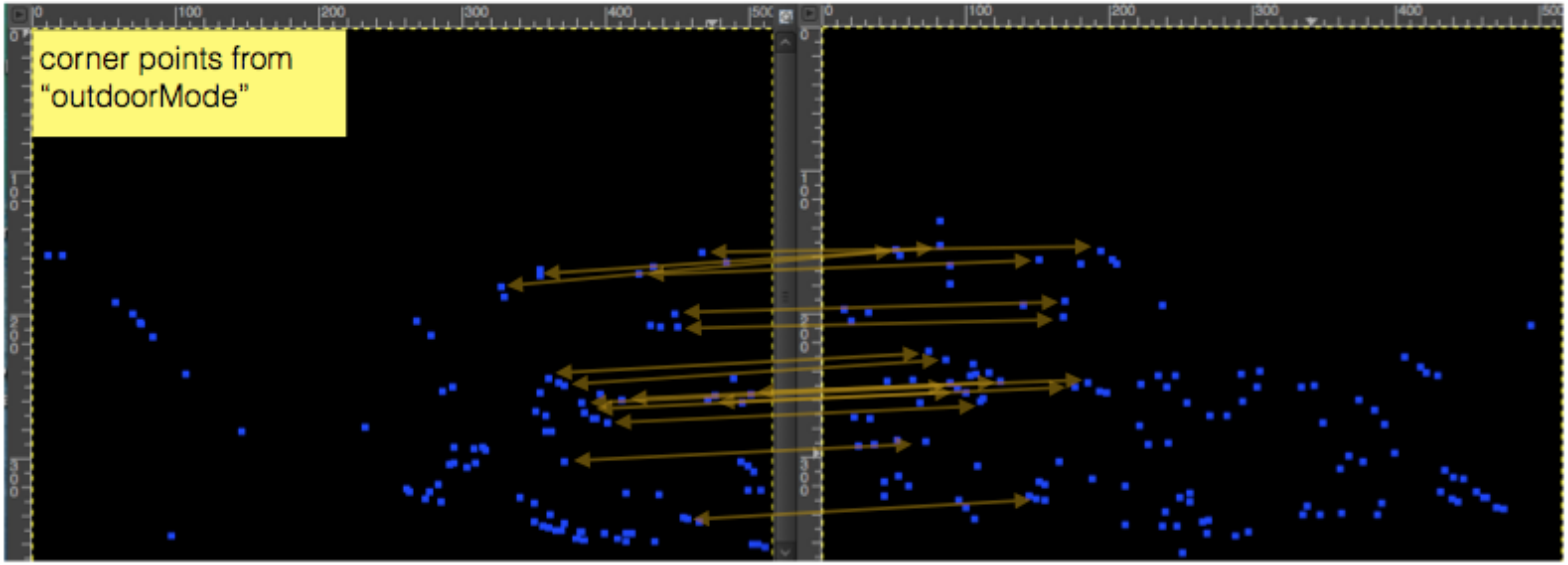
The Brown & Lowe 2003 images: point matching difficult because image intersection << difference



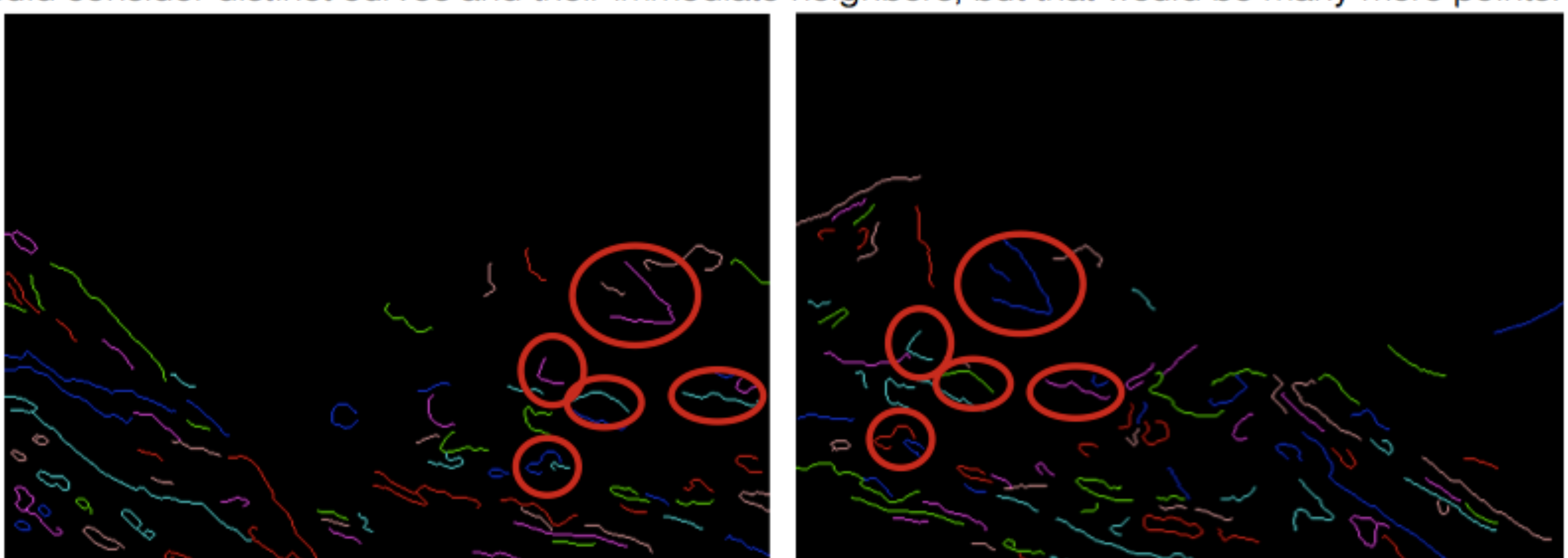
could consider distinct curves and their immediate neighbors, but that would be many more points:



The Brown & Lowe 2003 images: point matching difficult because image intersection < difference

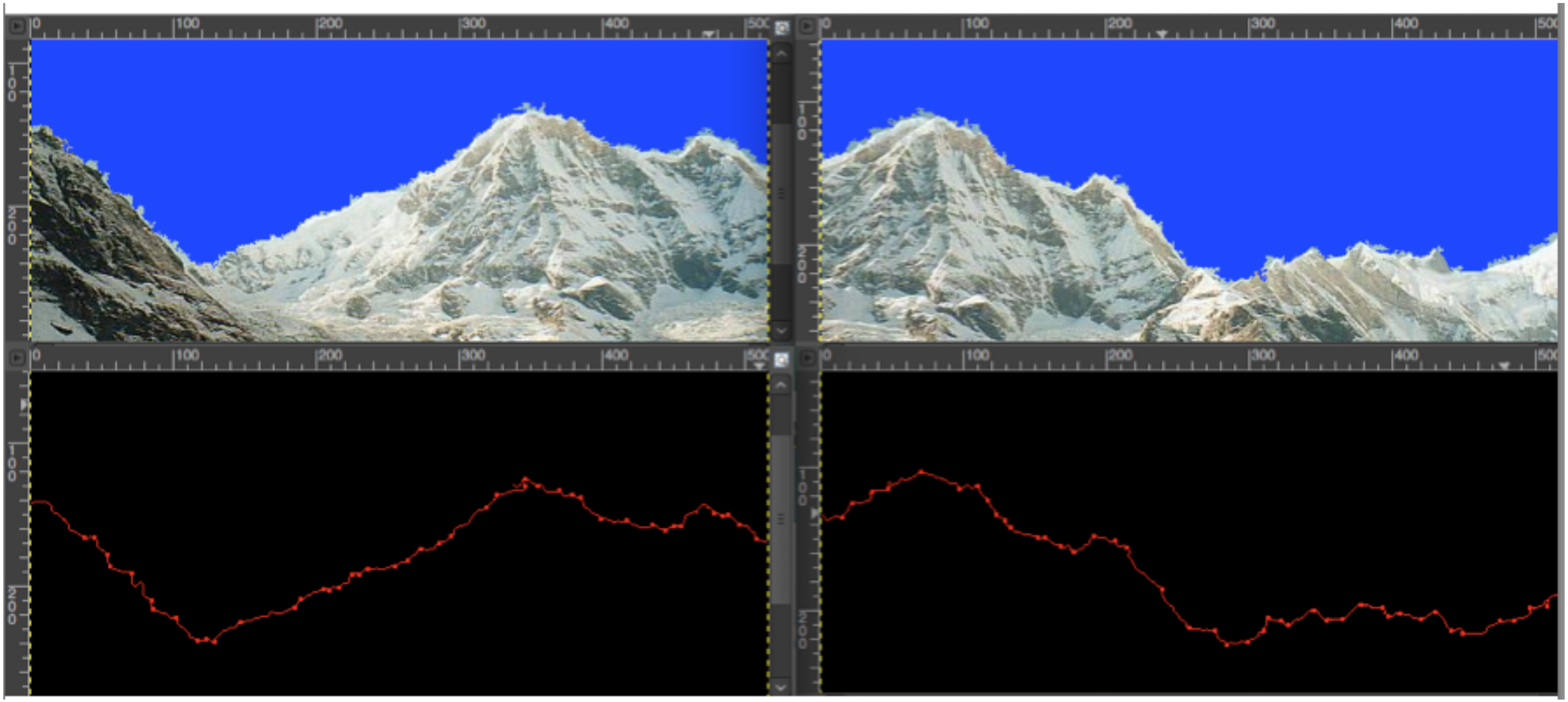


could consider distinct curves and their immediate neighbors, but that would be many more points:



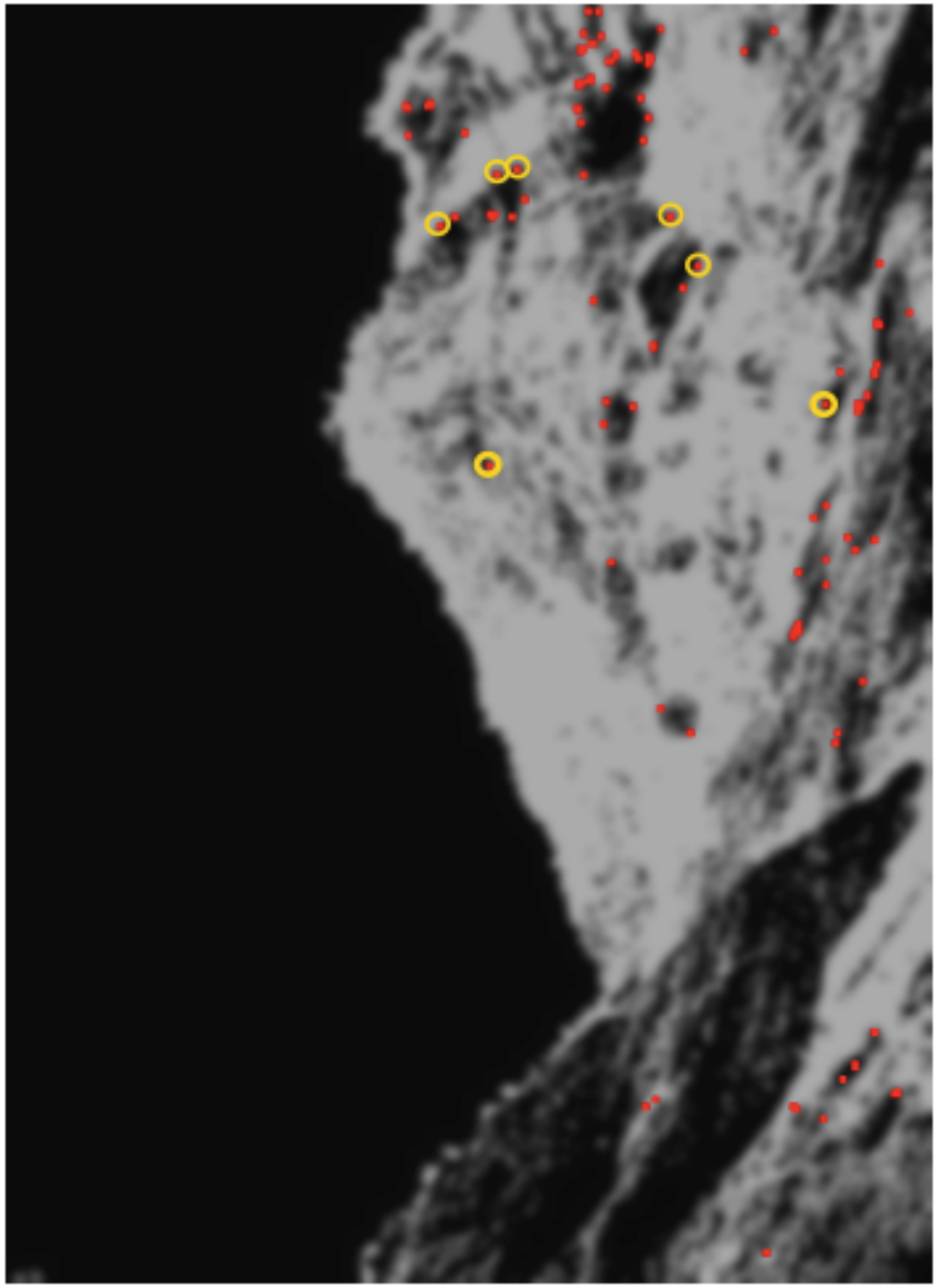
For the outdoor images, can find the sky and create a sky mask and also create corners from just the skyline.
(see skyline_extraction.pdf)

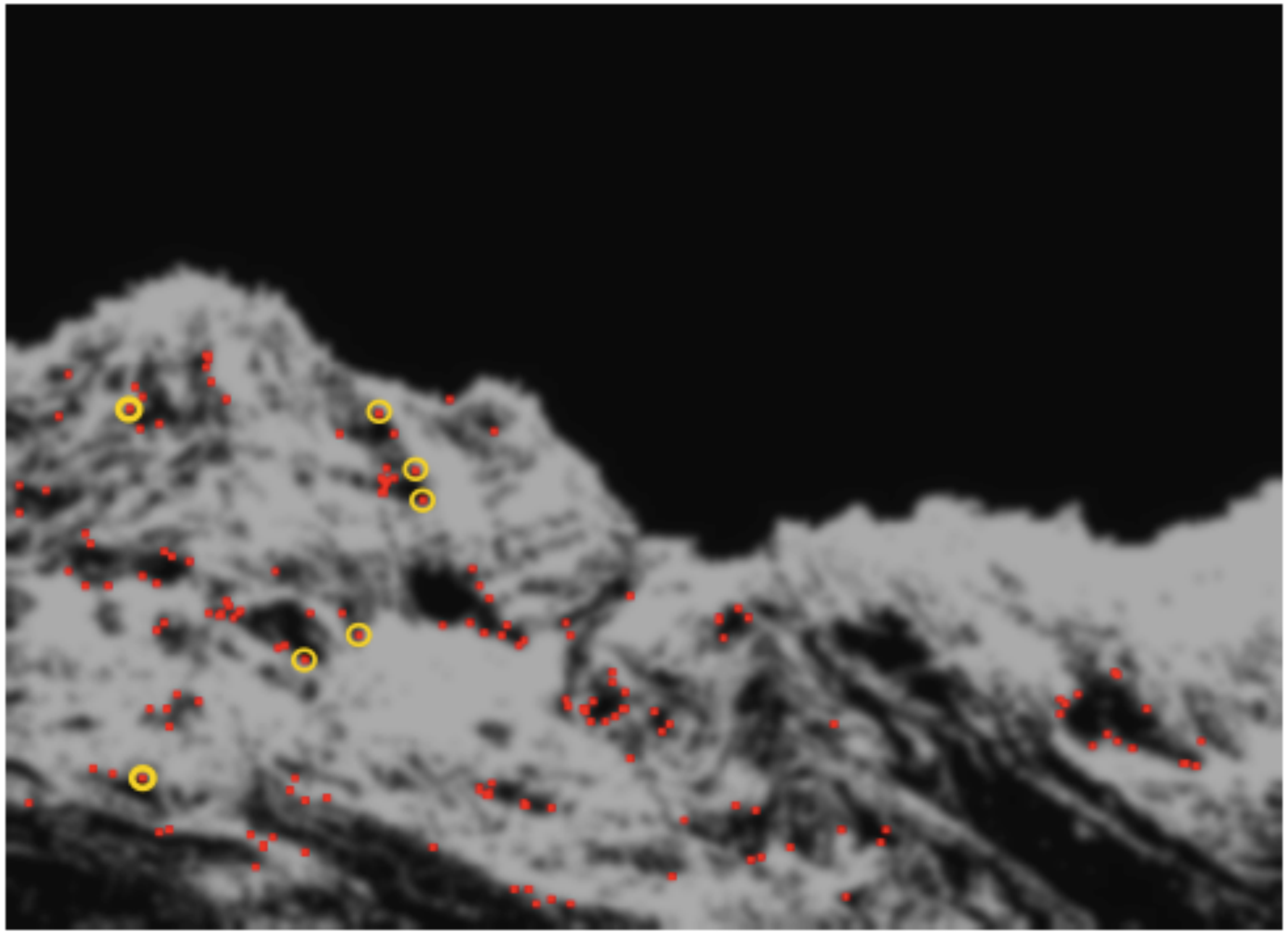
This sky mask helps to pre-process the image before feature finding and correspondence.



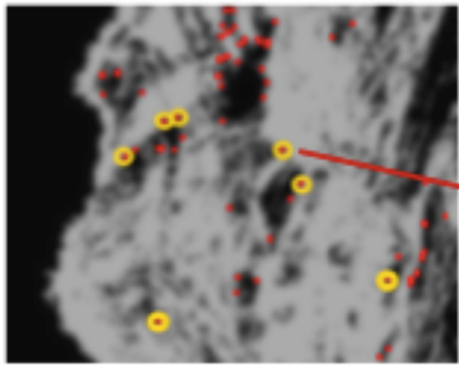
Need to match the corners using **<u>features</u>** because corner locations alone are often not enough to match correctly, especially when the number of truly matchable is smaller than the number of corners or when objects such as ridges exist giving large number of false matches to a different ridge of corners.

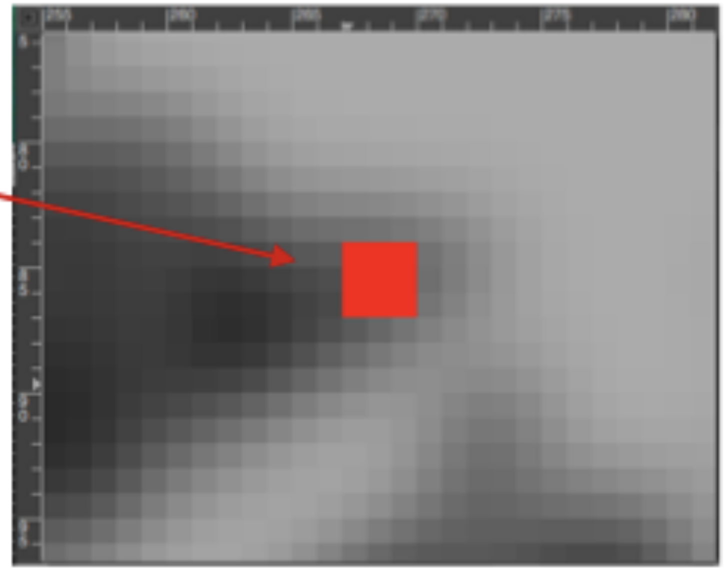
For features, need to determine an <u>orientation</u> of the corner region which is consistent w.r.t. the corner in any image. The highlighted corners are some examined in detail. The left is rotated to make sure the methods correctly handle rotation.

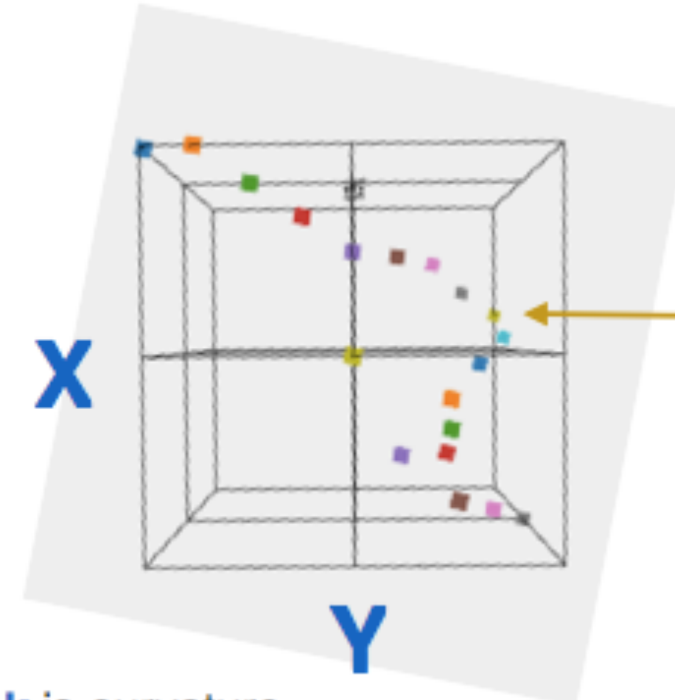




orientation of the feature: the edge already exists from making the corner, so can determine tangent at point.

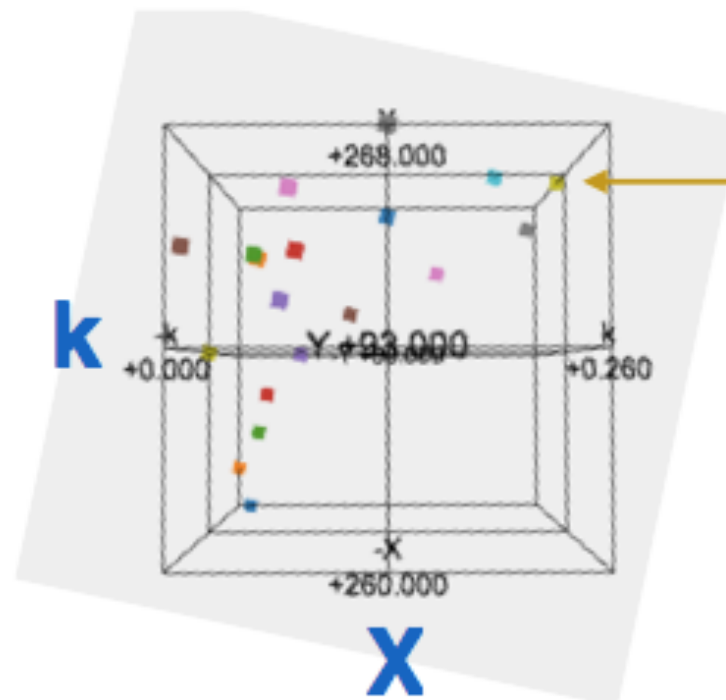






maximum in curvature

k is curvature

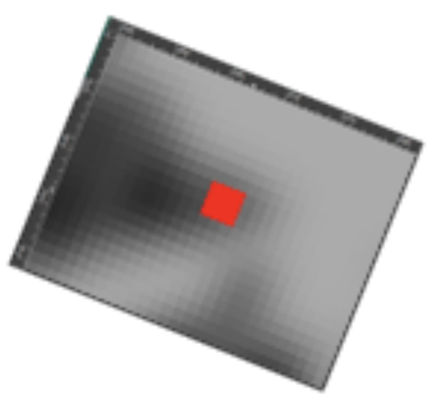


A vector perpendicular to k_max on this edge can be used to calculate the orientation of this region (where the region is defined by being a CSS corner).

```
deriving the perpendicular angle from the points directly to
the left and right of the maximum of curvature for points which
have curvature
k       x     y
0.24, 267, 84          dx,dy=(1,1), perp=45
0.26, 268, 85 <-max k
0.21, 268, 86          dx,dy=(0,1), perp==0

orientation = (45 + 0)/2. = 22.5
```

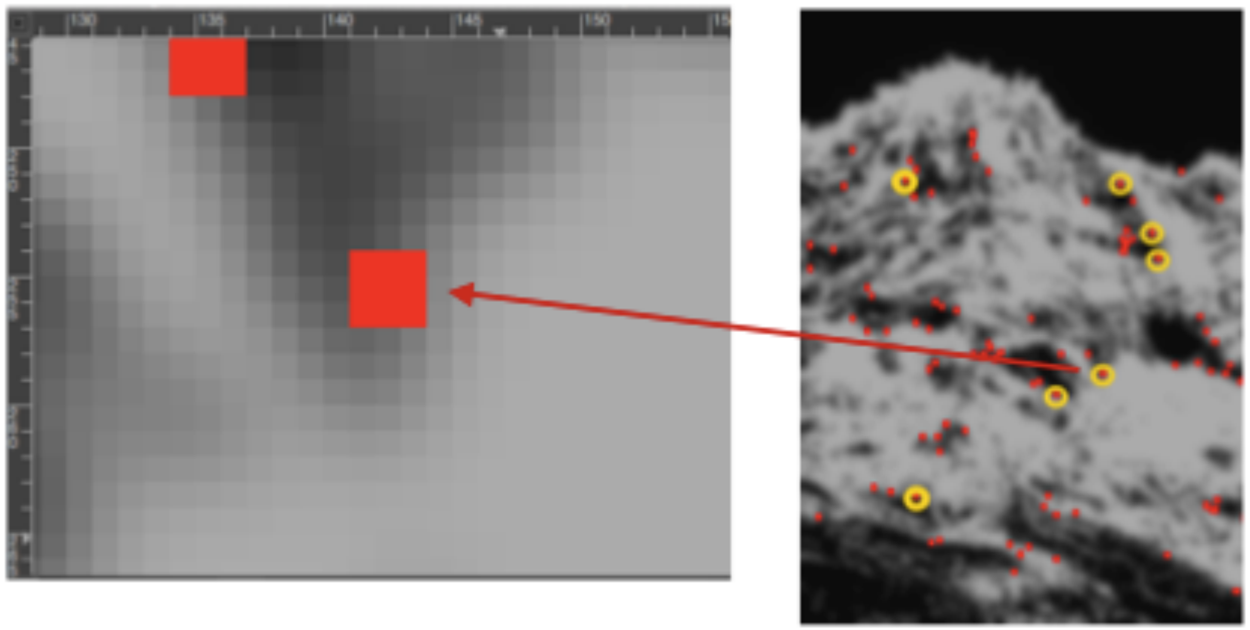
rotated



```
extract curvature
from the highest
resolution
scale space curves:

idx=164 (268.0,85.0)
k              x     y
0.08   for (258, 79)
0.04   for (259, 79)
0.01   for (260, 80)
0.00   for (261, 80)
0.02   for (262, 81)
0.03   for (263, 82)
0.06   for (264, 83)
0.10   for (265, 83)
0.17   for (266, 83)
0.24   for (267, 84)
0.26   for (268, 85)
0.21   for (268, 86)
0.13   for (267, 87)
0.04   for (266, 88)
0.04   for (266, 89)
0.07   for (266, 90)
0.06   for (265, 90)
0.00   for (266, 91)
0.07   for (267, 92)
0.13   for (268, 93)
0.15   for (268, 94)
```

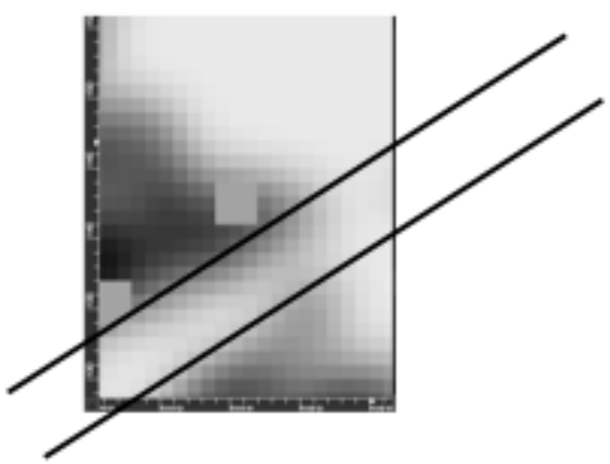
orientation.



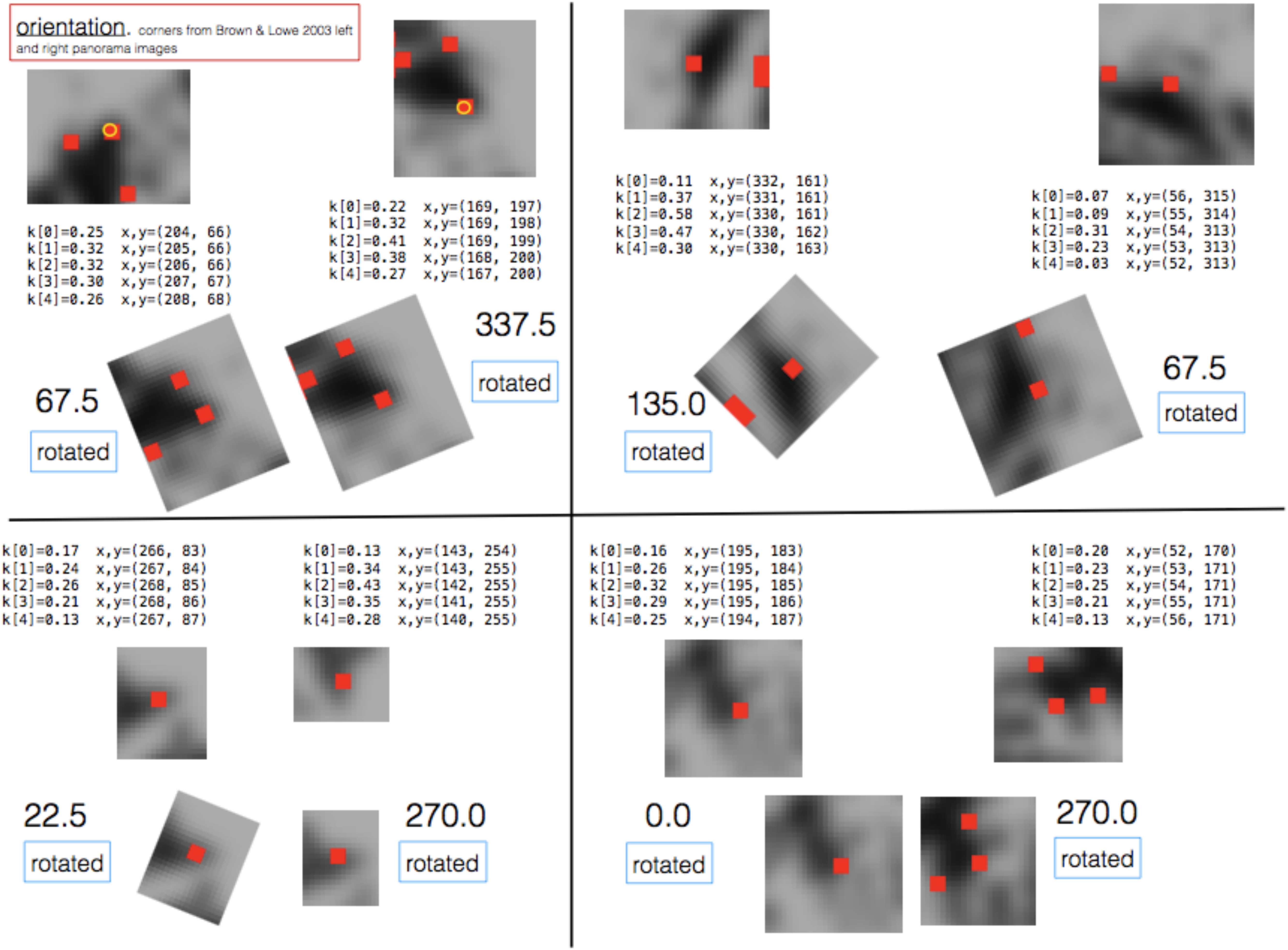
```
extract the final curvature from the highest resolution
scale space curves:
idx=112 (143.0,255.0)

k[110]=0.01  for (143, 253)
k[111]=0.13  for (143, 254)
k[112]=0.34  for (143, 255)  dx,dy=(-1,0)
k[113]=0.43  for (142, 255) <-- k_max
k[114]=0.35  for (141, 255)  dx,dy=(-1,0)
k[115]=0.28  for (140, 255)
k[116]=0.24  for (139, 255)
```

orientation =((-90)+(-90))/2. = -90



rotated

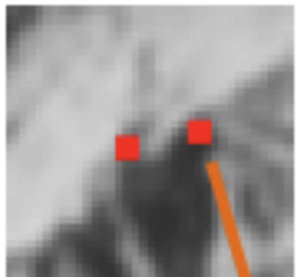

orientation. corners from Brown & Lowe 2003 left and right panorama images
k[0]=0.25 x,y=(204, 66)
k[1]=0.32 x,y=(205, 66)
k[2]=0.32 x,y=(206, 66)
k[3]=0.30 x,y=(207, 67)
k[4]=0.26 x,y=(208, 68)
k[0]=0.22 x,y=(169, 197)
k[1]=0.32 x,y=(169, 198)
k[2]=0.41 x,y=(169, 199)
k[3]=0.38 x,y=(168, 200)
k[4]=0.27 x,y=(167, 200)
67.5
rotated
337.5
rotated
k[0]=0.11 x,y=(332, 161)
k[1]=0.37 x,y=(331, 161)
k[2]=0.58 x,y=(330, 161)
k[3]=0.47 x,y=(330, 162)
k[4]=0.30 x,y=(330, 163)
k[0]=0.07 x,y=(56, 315)
k[1]=0.09 x,y=(55, 314)
k[2]=0.31 x,y=(54, 313)
k[3]=0.23 x,y=(53, 313)
k[4]=0.03 x,y=(52, 313)
135.0
rotated
67.5
rotated
k[0]=0.17 x,y=(266, 83)
k[1]=0.24 x,y=(267, 84)
k[2]=0.26 x,y=(268, 85)
k[3]=0.21 x,y=(268, 86)
k[4]=0.13 x,y=(267, 87)
k[0]=0.13 x,y=(143, 254)
k[1]=0.34 x,y=(143, 255)
k[2]=0.43 x,y=(142, 255)
k[3]=0.35 x,y=(141, 255)
k[4]=0.28 x,y=(140, 255)
22.5
rotated
270.0
rotated
k[0]=0.16 x,y=(195, 183)
k[1]=0.26 x,y=(195, 184)
k[2]=0.32 x,y=(195, 185)
k[3]=0.29 x,y=(195, 186)
k[4]=0.25 x,y=(194, 187)
k[0]=0.20 x,y=(52, 170)
k[1]=0.23 x,y=(53, 171)
k[2]=0.25 x,y=(54, 171)
k[3]=0.21 x,y=(55, 171)
k[4]=0.13 x,y=(56, 171)
0.0
rotated
270.0
rotated

orientation and junctions. corners

from Brown & Lowe 2003 left and right panorama images

```
k[0]=0.09  x,y=(204, 65)
k[1]=0.19  x,y=(205, 65)
k[2]=0.21  x,y=(206, 65)
k[3]=0.09  x,y=(206, 64)
k[4]=0.03  x,y=(206, 63)
```

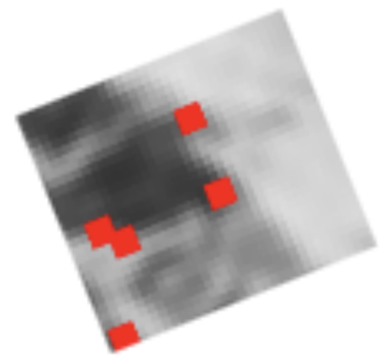
```
k[0]=0.21  x,y=(170, 198)
k[1]=0.32  x,y=(170, 199)
k[2]=0.41  x,y=(170, 200)
k[3]=0.38  x,y=(169, 201)
k[4]=0.27  x,y=(168, 201)
```

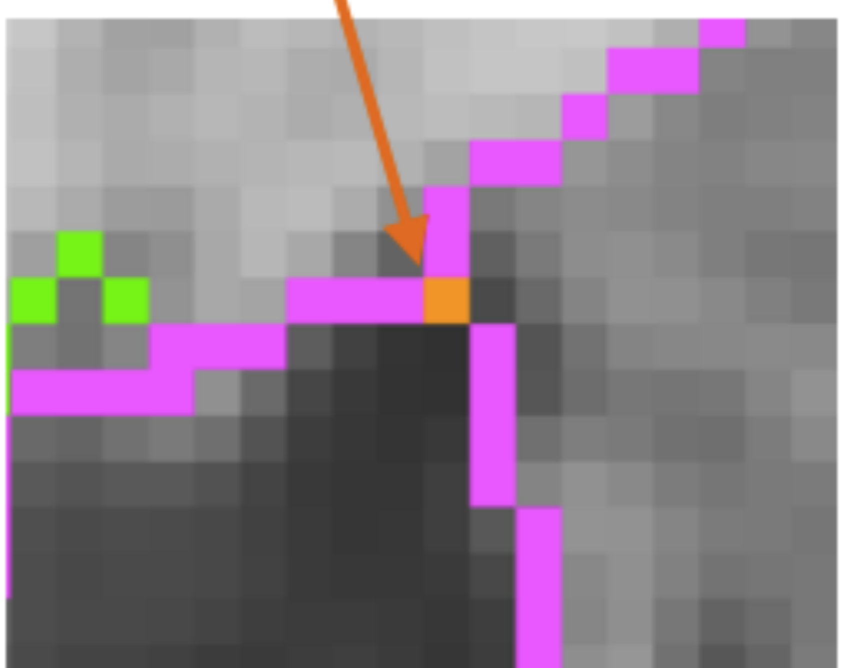


315.0

337.5

rotated



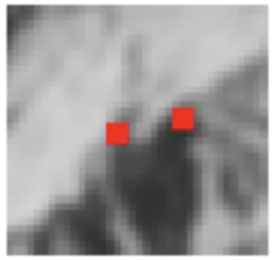


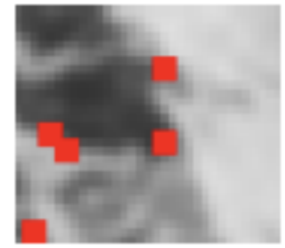
The left image corner is in a *junction* and that requires more complex analysis to determine here that the better edge for it is the lower right edge instead of the upper edge (the later gives an inconsistent orientation w.r.t. the right image).

Without the more complex analysis for now, will just create a CornerRegion for each combination and one of the 3 will match the right.

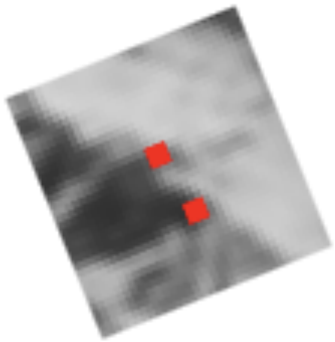
orientation and junctions (continued)

```
k[0]=0.02  x,y=(205, 65)
k[1]=0.21  x,y=(206, 65)
k[2]=0.02  x,y=(207, 66)
```





```
k[0]=0.21  x,y=(170, 198)
k[1]=0.32  x,y=(170, 199)
k[2]=0.41  x,y=(170, 200)
k[3]=0.38  x,y=(169, 201)
k[4]=0.27  x,y=(168, 201)
```

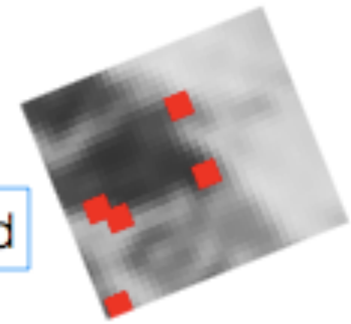


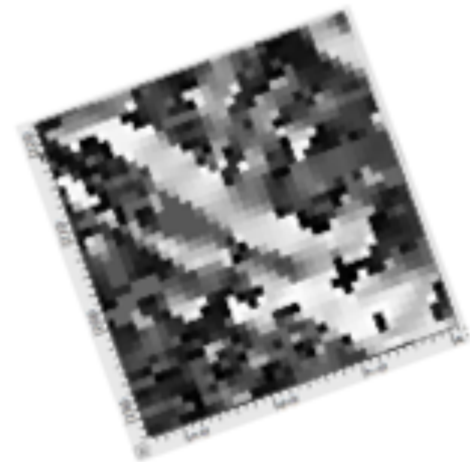
67.5

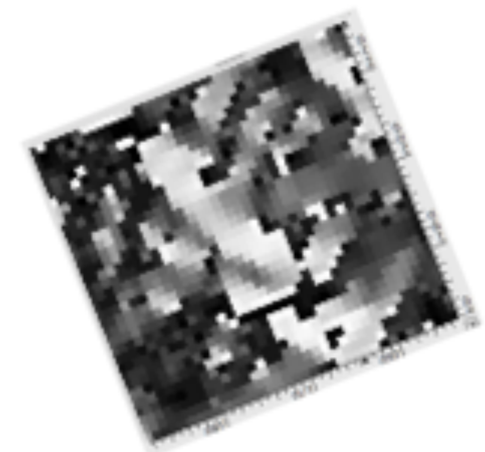
rotated

337.5

rotated



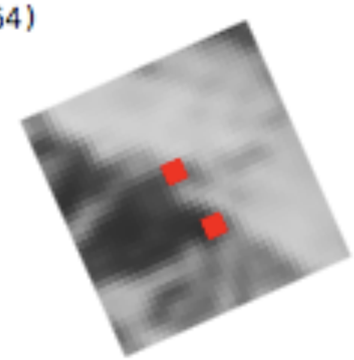


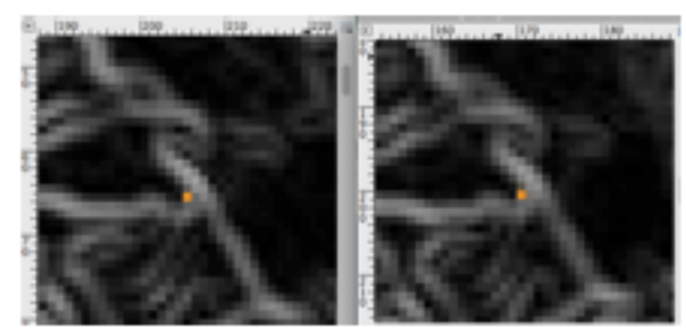


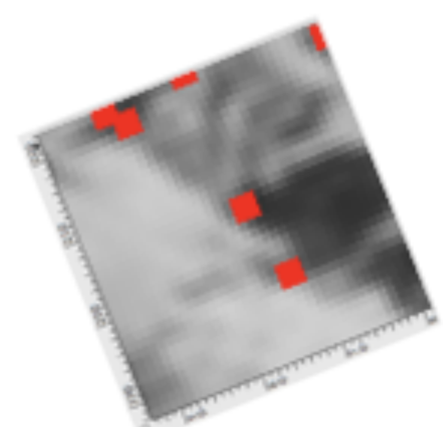
```
k[0]=0.02  x,y=(205, 65)
k[1]=0.21  x,y=(206, 65)
k[2]=0.02  x,y=(206, 64)
```

66

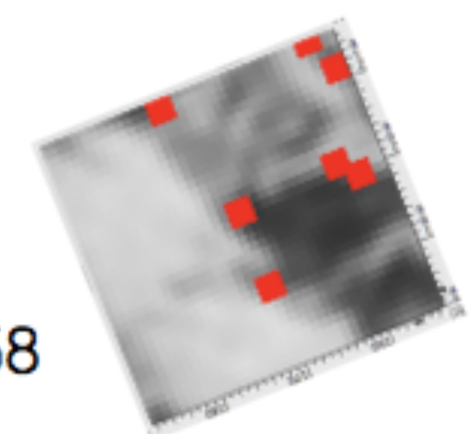
rotated

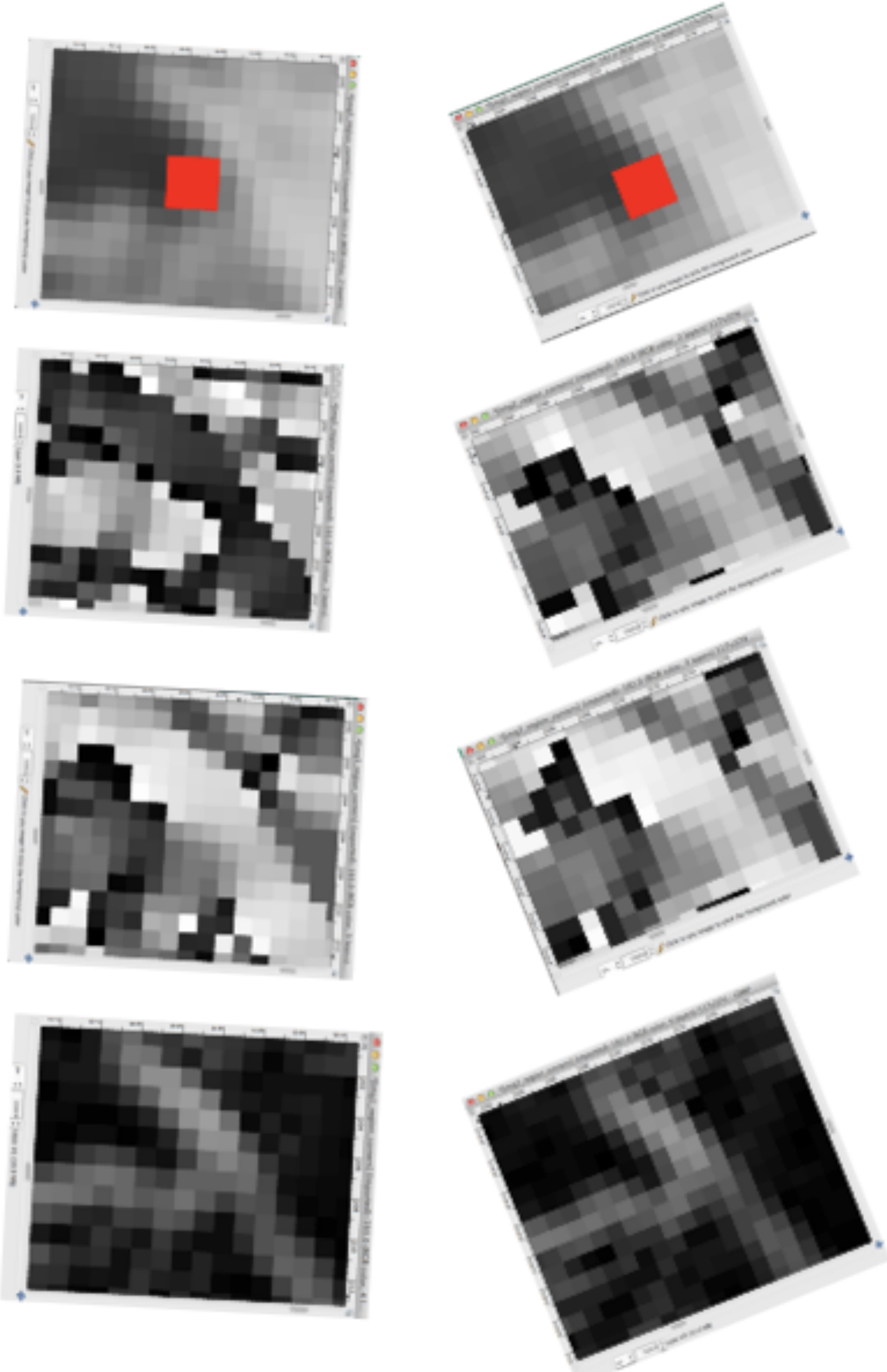






250 158





93 338

greyscale rotated to “dominant orientation”

gradient theta rotated to “dominant orientation”

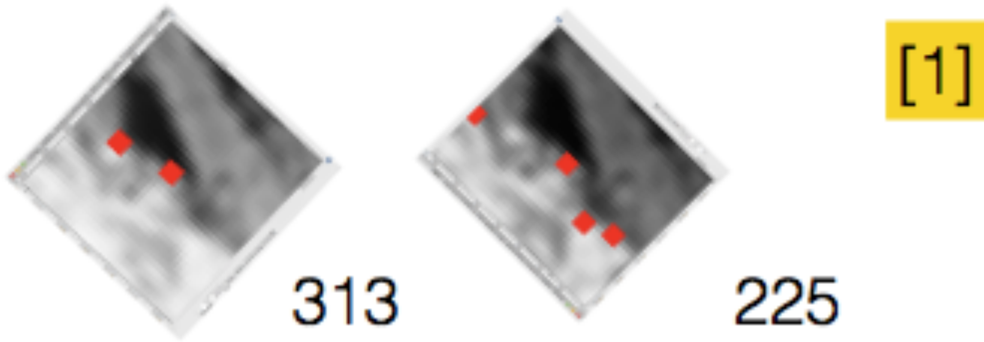
gradient theta rotated to “dominant orientation” with “dominant orientation” subtracted from theta

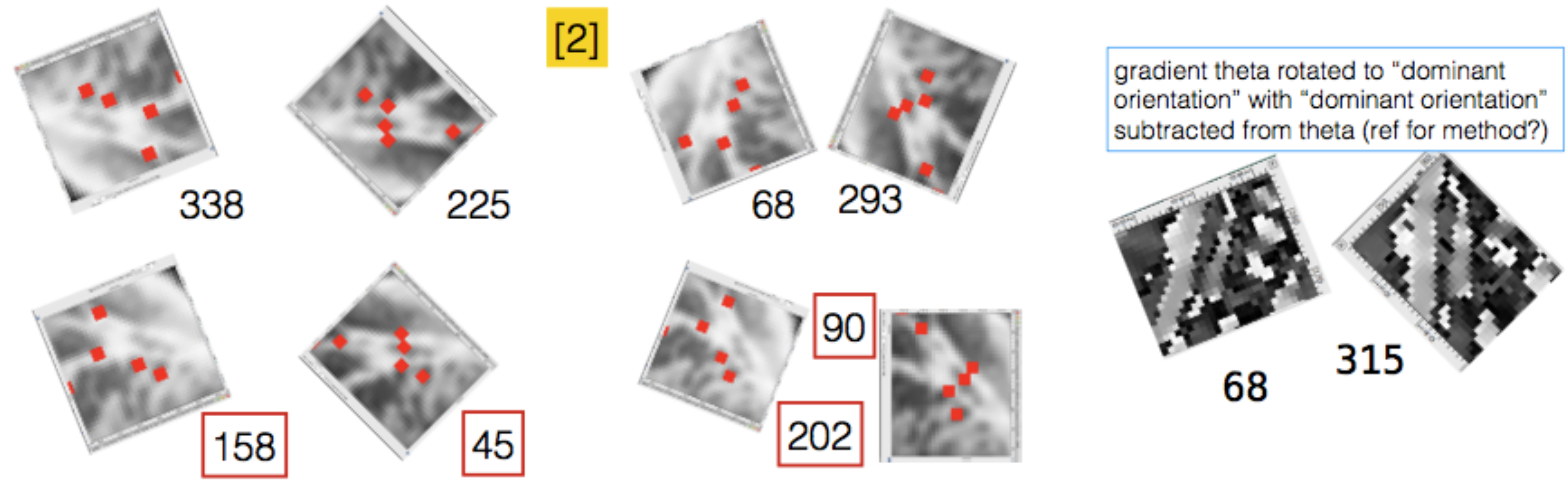
can see that image 2 should have preferred match w/ orientation 67.5 instead of 93

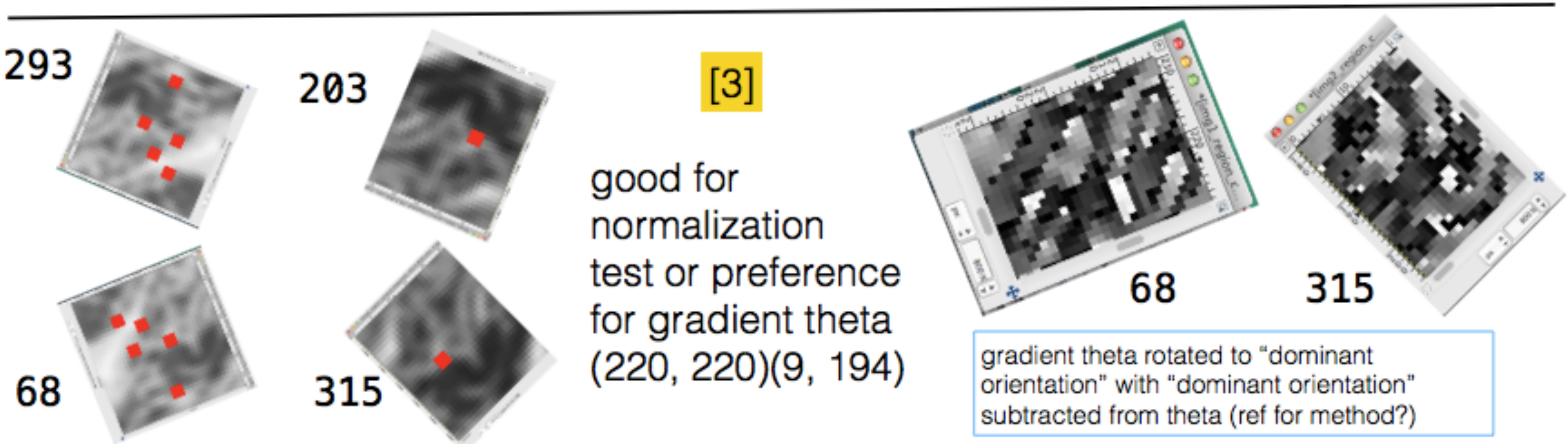
can see that SSD on gradient theta could be a good descriptor with the orientation corrections and quadrant consideration (e.g. 0 diff 350 is 10, not 350)

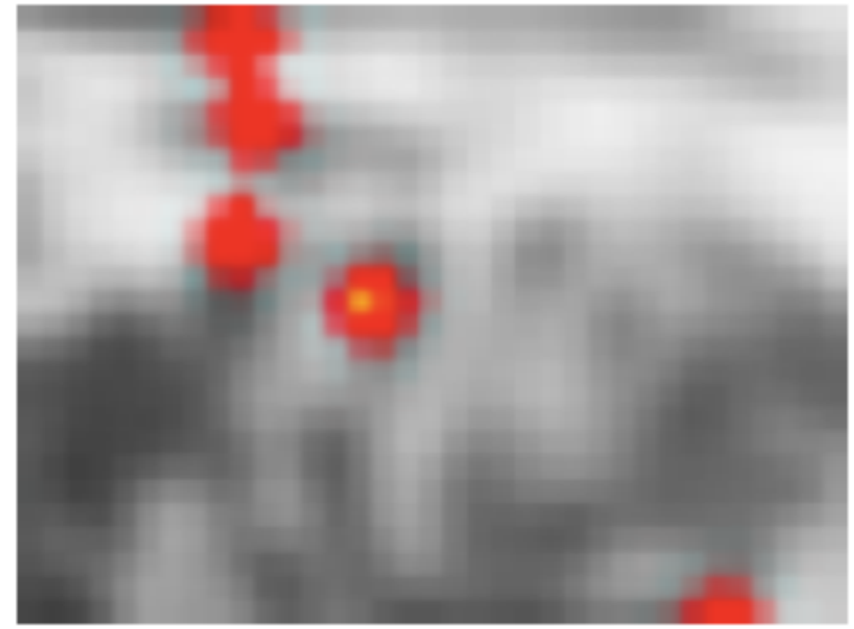
can see that SSD on gradient could be a good descriptor but may not be as tolerant of skew as subcells of histograms of orientation?

will use gradient, but binned into 2x2 cells, and 16 of them surrounding the corner.


[1]
313
225


[2]
338
225
68
293
gradient theta rotated to "dominant orientation" with "dominant orientation" subtracted from theta (ref for method?)
158
45
90
202
68
315

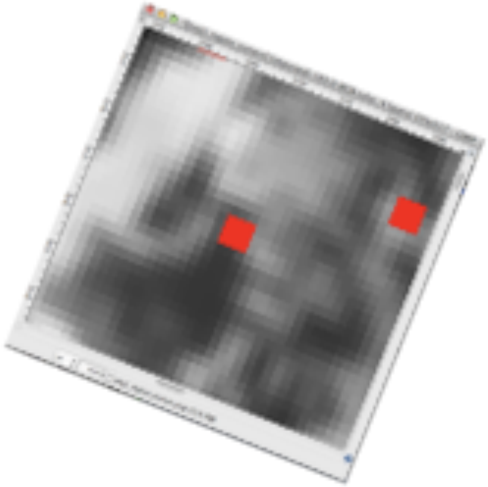

293
203
[3]
good for normalization test or preference for gradient theta (220, 220)(9, 194)
68
315
68
315
gradient theta rotated to "dominant orientation" with "dominant orientation" subtracted from theta (ref for method?)



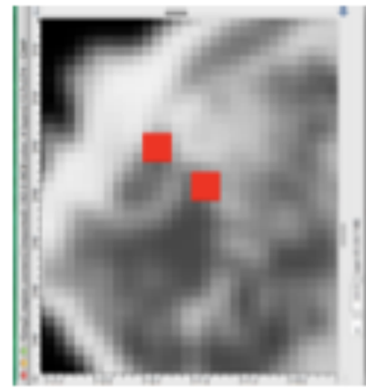
left and right clips from the Brown & Lowe showing projection and illumination differences.

Binning the intensity descriptor pixels into 2x2 cells and subtracting the mean of the descriptor intensity values from itself helps to remove the illumination differences and projection differences for those within that small range.

Note that the rectangular radius around the central pixel is somewhat large, 12 pixels (note too that the snapshot above is larger than a descriptor's region).

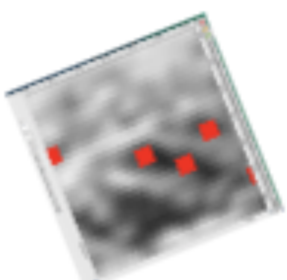


0 or 22.5

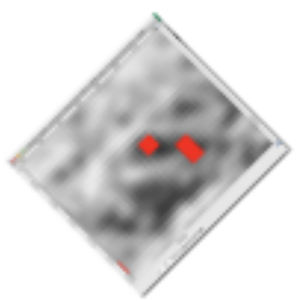


270

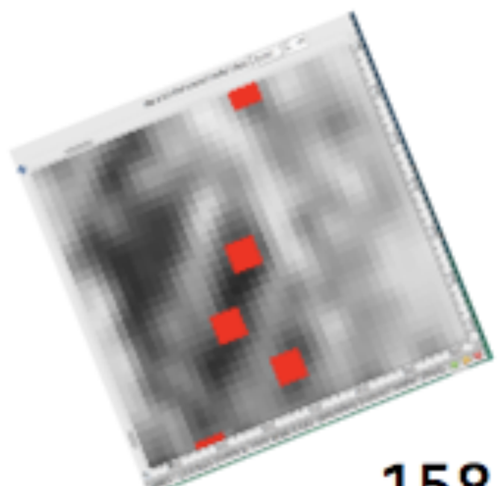
[4]



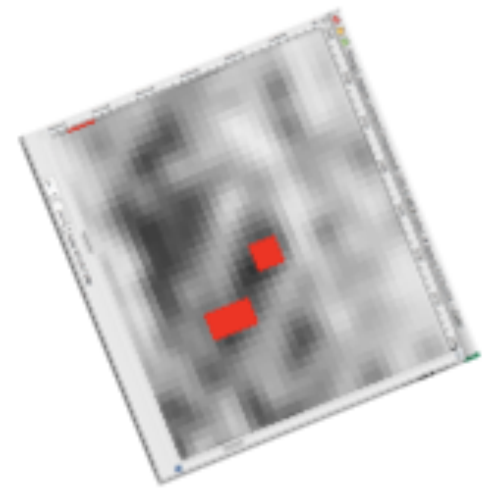
68



315

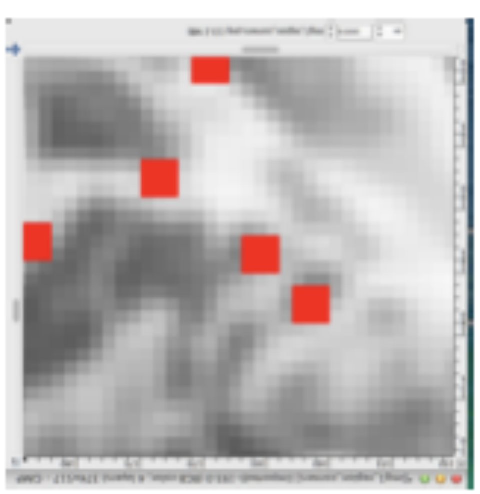


158

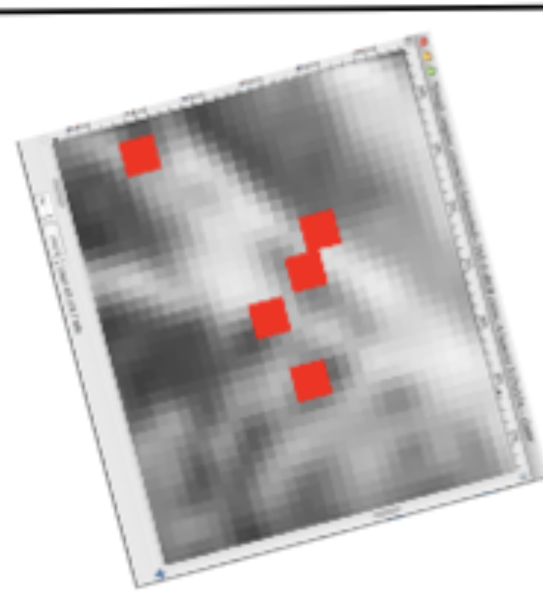


68

[5]



180



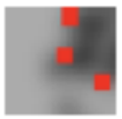
75?

[6]

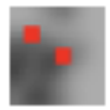
orientation. corners from Venturi dataset

```
k[0]=0.15  x,y=(461, 449)
k[1]=0.23  x,y=(460, 448)
k[2]=0.29  x,y=(459, 447)
k[3]=0.26  x,y=(459, 446)
k[4]=0.16  x,y=(459, 445)
```

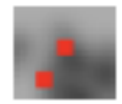
```
k[0]=0.13  x,y=(157, 466)
k[1]=0.19  x,y=(156, 466)
k[2]=0.22  x,y=(155, 466)
k[3]=0.20  x,y=(154, 466)
k[4]=0.16  x,y=(153, 467)
```



202.5



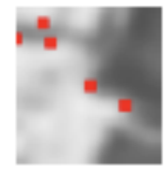
90.0



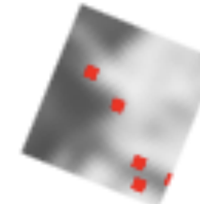
rotated

```
k[0]=0.18  x,y=(293, 496)
k[1]=0.24  x,y=(292, 495)
k[2]=0.29  x,y=(291, 494)
k[3]=0.25  x,y=(291, 493)
k[4]=0.13  x,y=(291, 492)
```

```
k[0]=0.18  x,y=(105, 298)
k[1]=0.33  x,y=(105, 297)
k[2]=0.36  x,y=(106, 296)
k[3]=0.26  x,y=(107, 296)
k[4]=0.14  x,y=(108, 296)
```



202.5

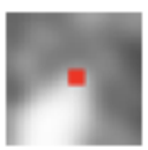


112.5

rotated

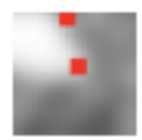
```
k[0]=0.27  x,y=(449, 222)
k[1]=0.33  x,y=(448, 221)
k[2]=0.37  x,y=(447, 220)
k[3]=0.35  x,y=(446, 220)
k[4]=0.26  x,y=(445, 220)
```

```
k[0]=0.26  x,y=(382, 448)
k[1]=0.29  x,y=(383, 447)
k[2]=0.31  x,y=(384, 446)
k[3]=0.29  x,y=(384, 445)
k[4]=0.23  x,y=(384, 444)
```

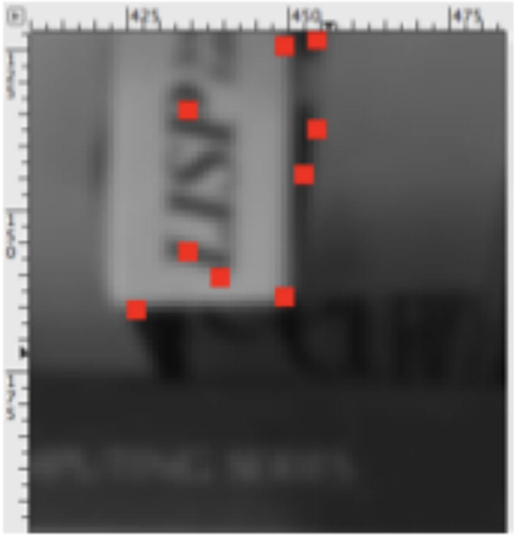


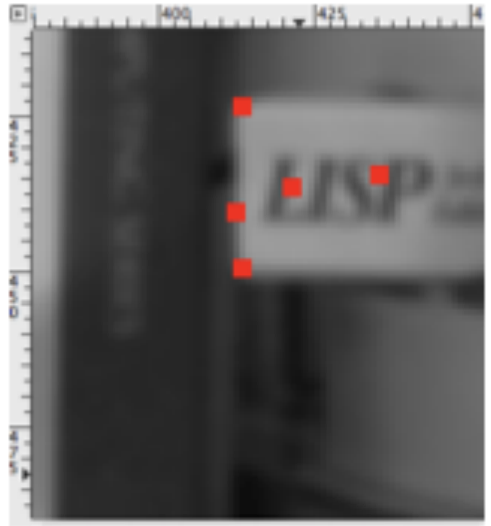
67.5

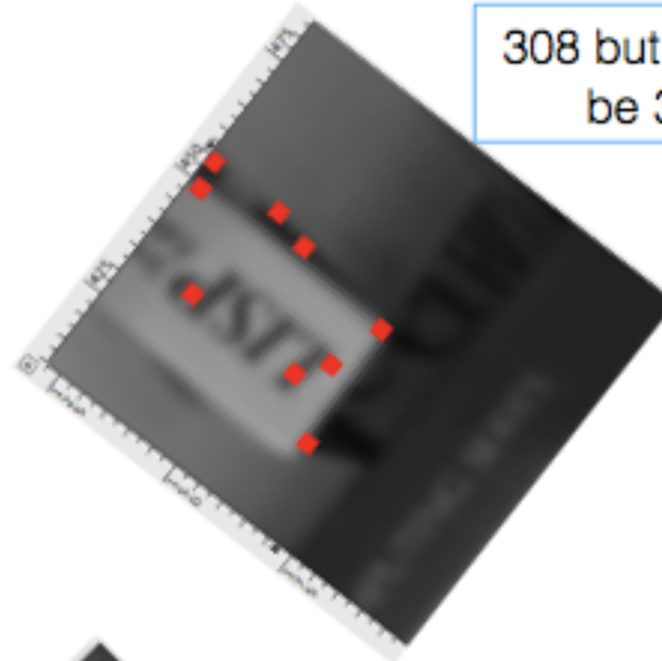
337.5



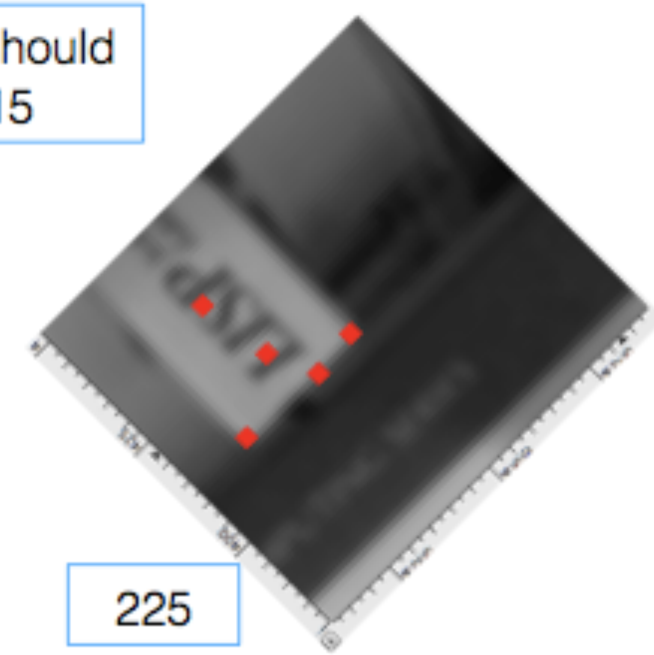
rotated



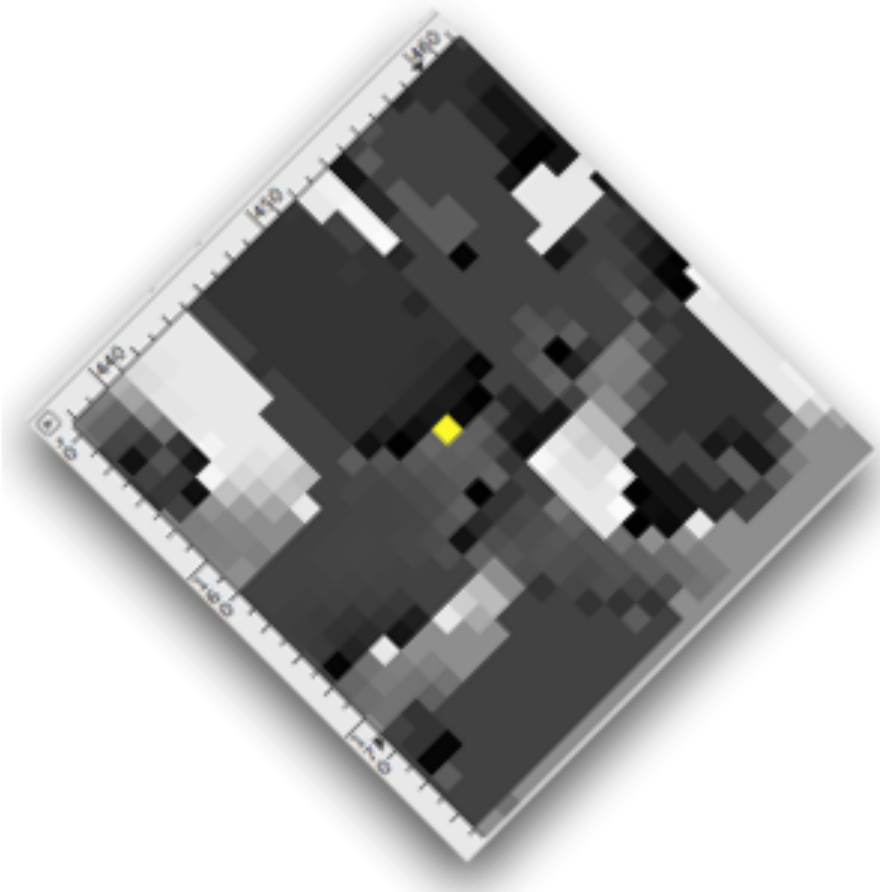


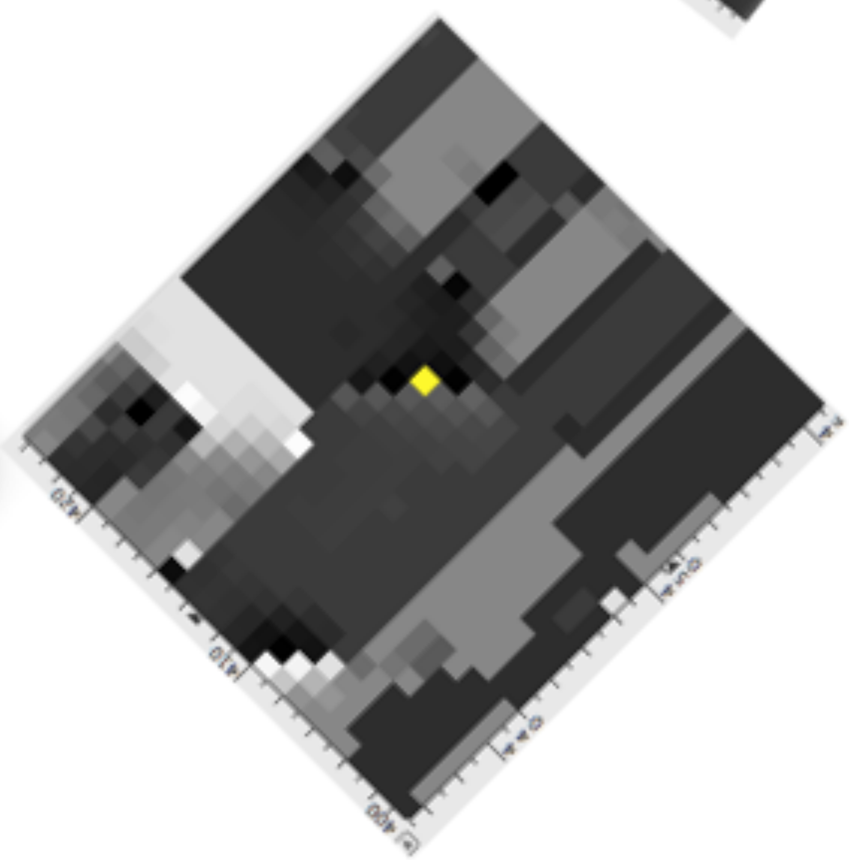


308 but should be 315



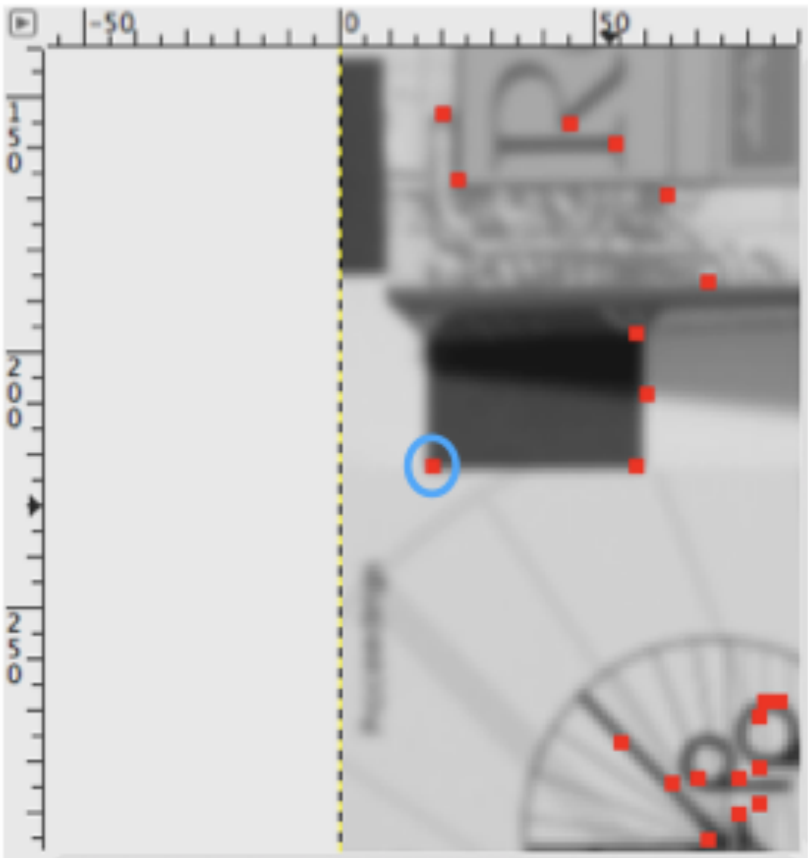
225

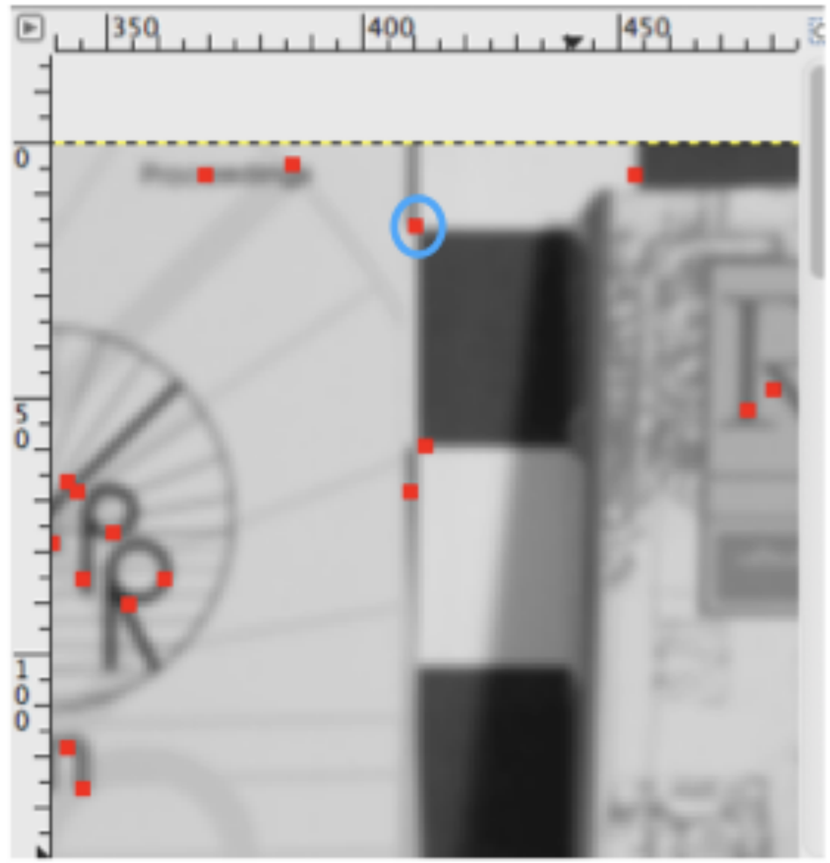




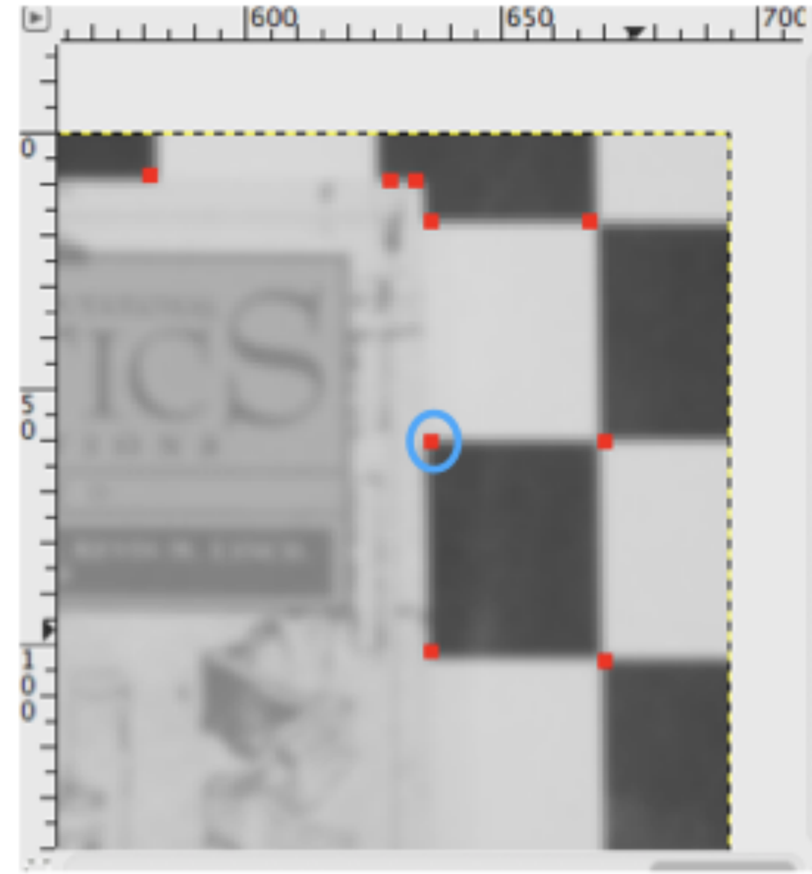
projection shows a different neighboring region for the book. so theta is very good for identifying the book consistently, but object needs to be on an unvarying background/ foreground or one should only compare theta within the contour (bounds) of the book.

the theta descriptor SSD is larger than the error from auto-correlation. discard in favor of clearer matches and note for a future algorithm which recognizes contours and matches within…





true match (not found)



false match (found)

this is an example where a repeated pattern is better matched elsewhere in the image due to 2 things: (1) it’s a pattern (very big texture) and (2) the background/foreground changes near the corner change the appearance enough that the true match isn’t found.

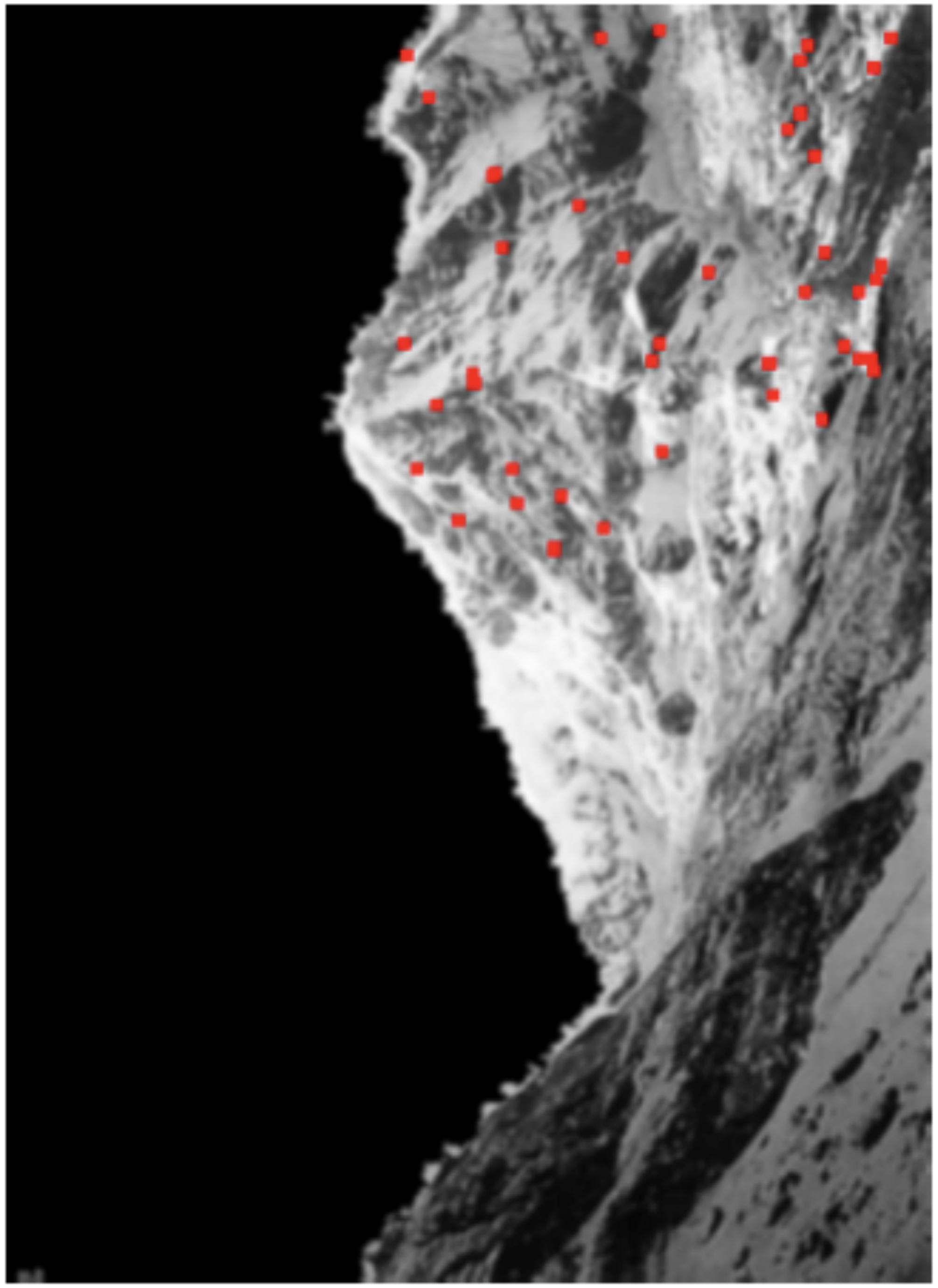
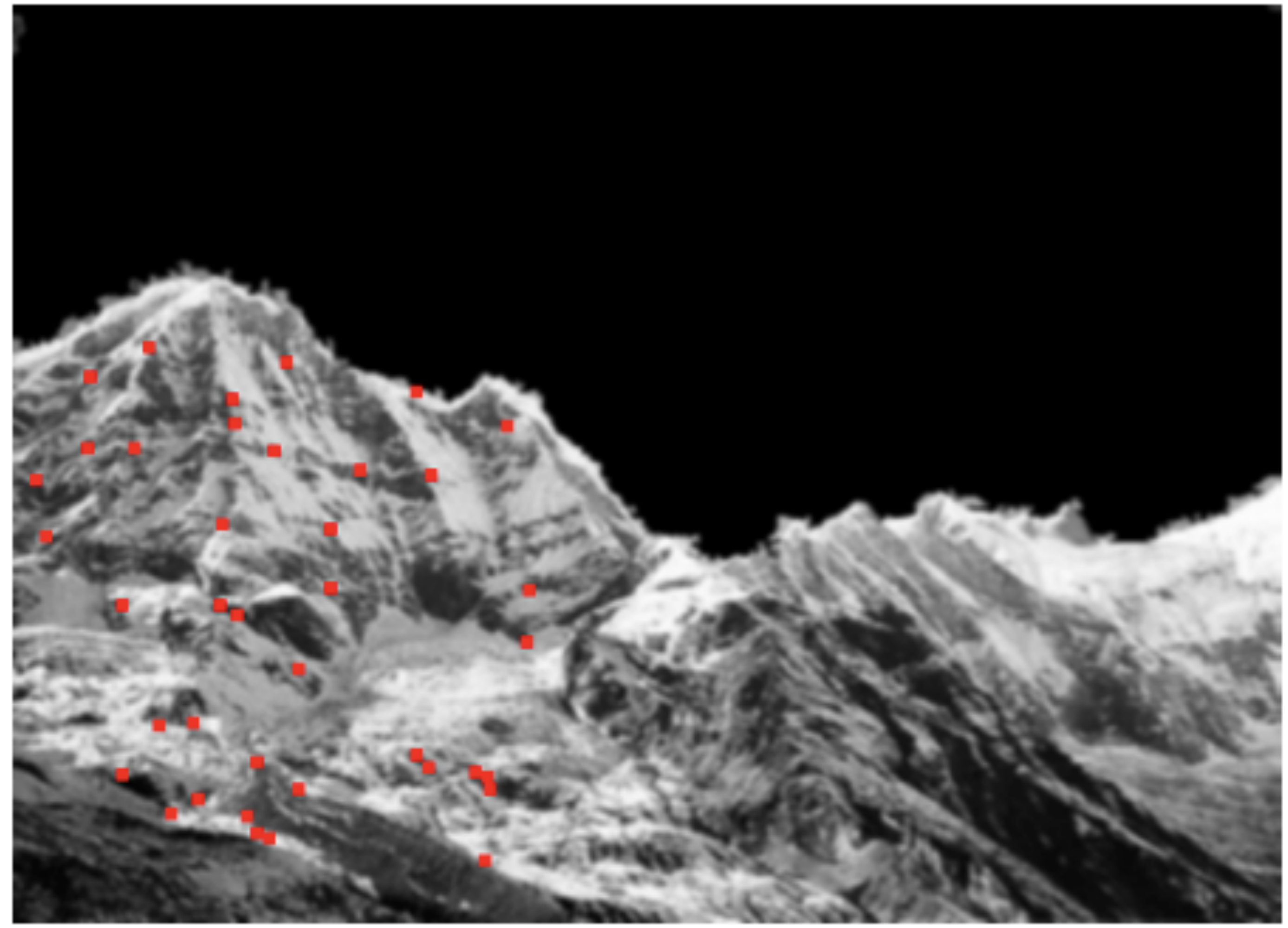
where “found” is referring to the matching algorithm to make rough correspondence lists before use of the epipolar projection solver.

might need to consider the top k matches, especially if the image is known to contain repeated patterns/textures.

Euclidean transformation from pairwise calc of known points:
rotationInRadians=4.5554905
rotationInDegrees=261.01037890524134 scale=1.004318
translationX=266.80643
translationY=5.852234 originX=0.0 originY=0.0

Feature matches of corner regions followed by a look at the implied rotations by frequency then filtering with that to see the most frequent translation in X then filtering by that to find the most frequent translation in Y:

solution
rotation=270.0+-20
translationX=215.0+-55.84706878621094
translationY=-25.0+=29.85197257995605

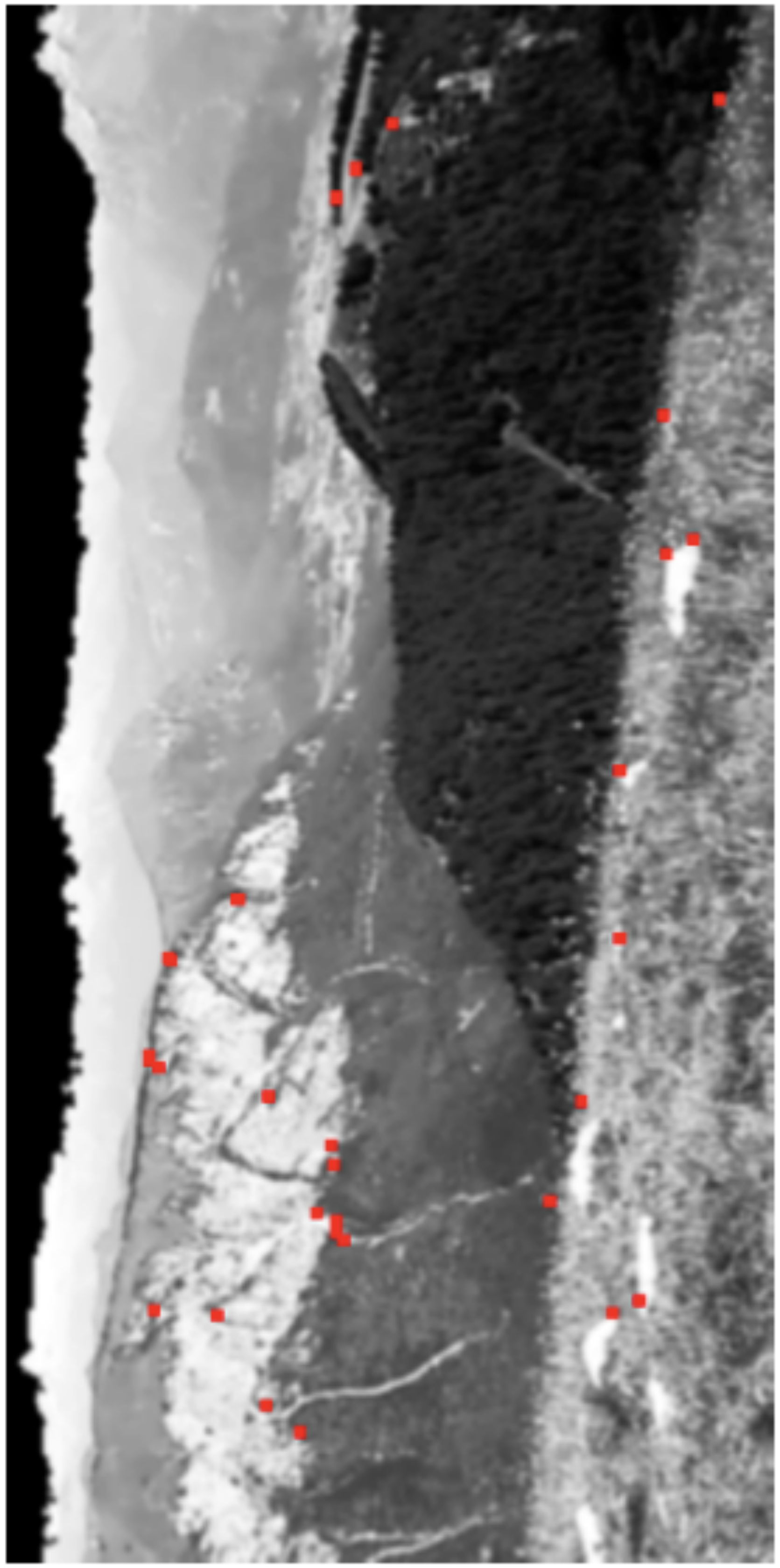
 

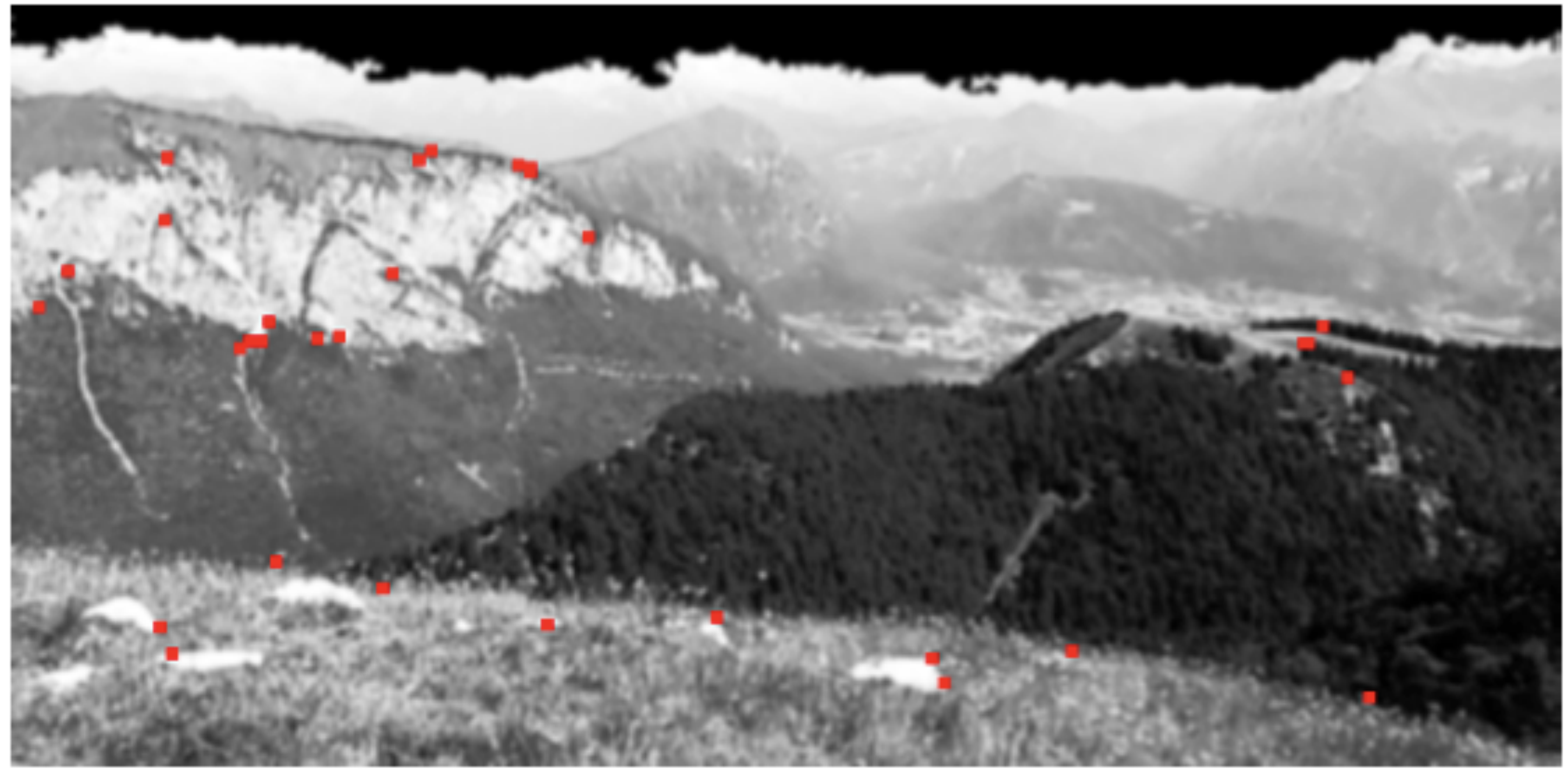
result is a list of correspondence, usable as input to the epipolar solver which uses RANSAC to discard outliers while solving the fundamental matrix.

Euclidean transformation from pairwise
calc of known points:
params=rotationInRadians=4.712166
rotationInDegrees=269.9872145691191 scale=1.0279127
translationX=614.1122 translationY=-4.460491
originX=0.0 originY=0.0

Feature matches of corner regions followed by a look at the implied rotations by frequency then filtering with that to see the most frequent translation in X then filtering by that to find the most frequent translation in Y:

solution
rotation=280.0+-20
translationX=625.0+-31.55487060546875
translationY=5.0+=0.0





result is a list of correspondence, usable as input to the epipolar solver which uses RANSAC to discard outliers while solving the fundamental matrix.

Euclidean transformation from pairwise
calc of known points:
params=rotationInRadians=4.512667
rotationInDegrees=258.556783710076
scale=1.0265783
translationX=639.77747 translationY=12.565971
originX=0.0 originY=0.0

Feature matches of corner regions followed by a look at the implied rotations by frequency then filtering with that to see the most frequent translation in X then filtering by that to find the most frequent translation in Y:

solution
rotation=270.0+-20
translationX=625.0+-55.98248291015625
translationY=25.0+=44.471153259277344

On already rectified images, the rough euclidean solution that is used to remove more distant matched pairs probably needs a 2nd order term, at least for translations.

Have algorithm to match features, but still need to know scale between the images.
For many sets this is already known to be ‘1’, but for images of different cameras, etc, it isn’t known.
A scale between both images can be found using scale space, but that still requires identifying similar objects in both images or a similar population of objects.
With CSS, can make scale space maps with inflection points for a large range of Gaussian sigma to find the peaks of the inflection points in the scale space maps. The same point’s sigma divided by the same in the other image’s scale space maps gives scale (and location of those points further gives rotation and translation with 2 sets of points).

In order to use scale space to solve for scale, need to make sure that the comparison is among same object population or specifically identify the objects with methods such as in the ContourMatcher, ContourFinder and InflectionMapper (see doc/contours.pdf too).

Because edges don’t necessarily by default in this project create contours (closed curves enclosing objects of interest), will find significant blobs and create contours for them as ordered boundary pixels on the blob perimeters (concave hulls). Then will find the inflection points on those and create scale space maps, and use the contour matcher to solve for the scale.   This looks to be feasible with the start images binned to a size < 200 x 200 and segmentation somewhere between k=2 and k = 5. see next…

Blob finding for scale

— perform histogram equalization if mean is too far from median or the two images are too different for those stats.

— color segmentation of ***k=2*** to reduce image to 2 bands of intensities.

— contiguous pixel group finder for each of the 2 bands using point limits of: smallestGroupLimit = ***100***; largestGroupLimit = ***5000***

— implemented a blob ordered perimeter extractor to create closed curves (concave hulls) usable as contours for matching.

The blob finding performed on images binned to < 300 x 300 and on the full images seems to produce feasible matching features even though there are few.

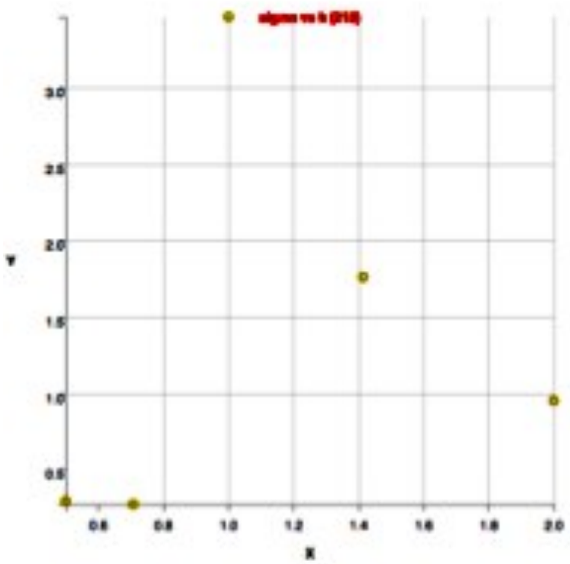
The blob contours are used to solve for scale and to roughly solve for rotation and translation.

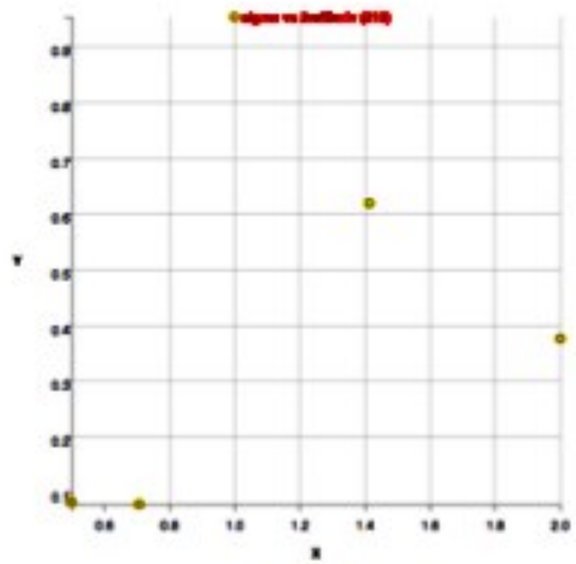
Once scale is calculated from blobs, ***features*** are used to create correspondence lists for input to epipolar projection solver (see previous pages).

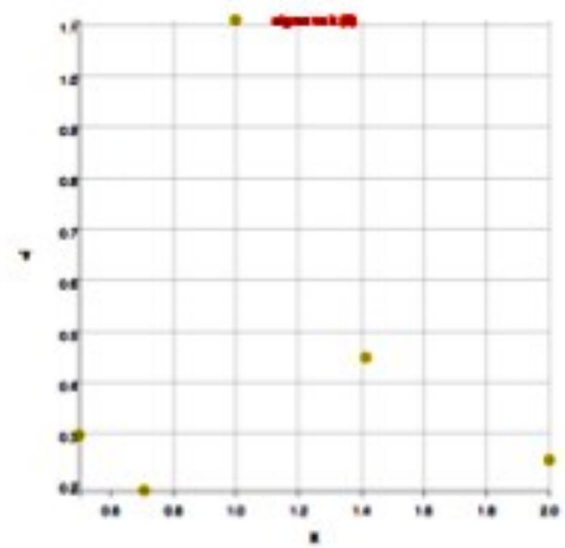
As a tangent, a quick look at “characteristic scale” from 2nd derivatives and curvature of corners in scale space maps.
The 2 images of the same objects are from the same camera and same camera settings, so the responses are similar. Shown are not the full range to sigma=256.

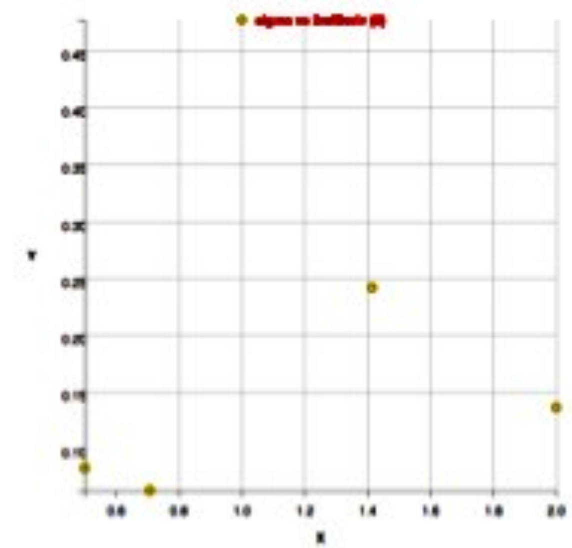
X’ is gaussian first derivative, X’’ is gaussian second derivative. t is unit along path in scale space. o~ is gaussian sigma.

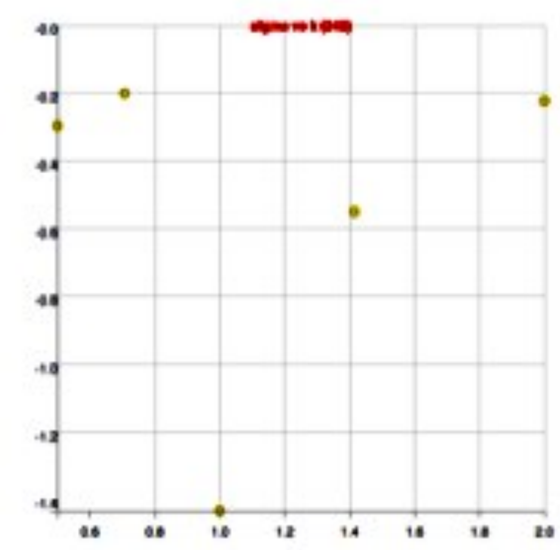
$$k(t,o\sim) = \frac{X'(t,o\sim) * Y''(t,o\sim) - Y'(t,o\sim) * X''(t,o\sim)}{(X'^2(t,o\sim) + Y'^2(t,o\sim))^{1.5}}$$

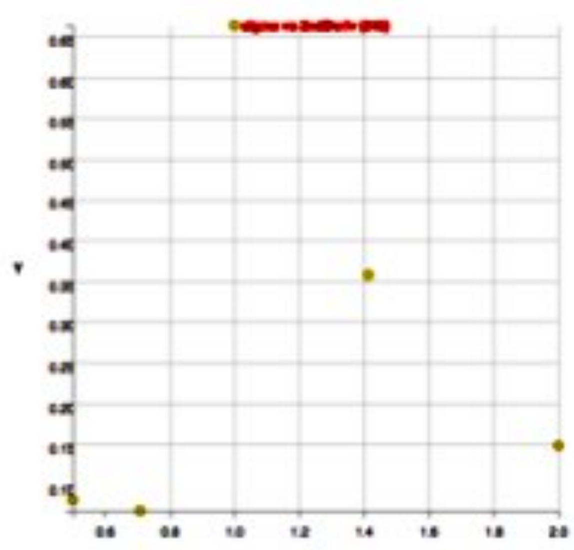






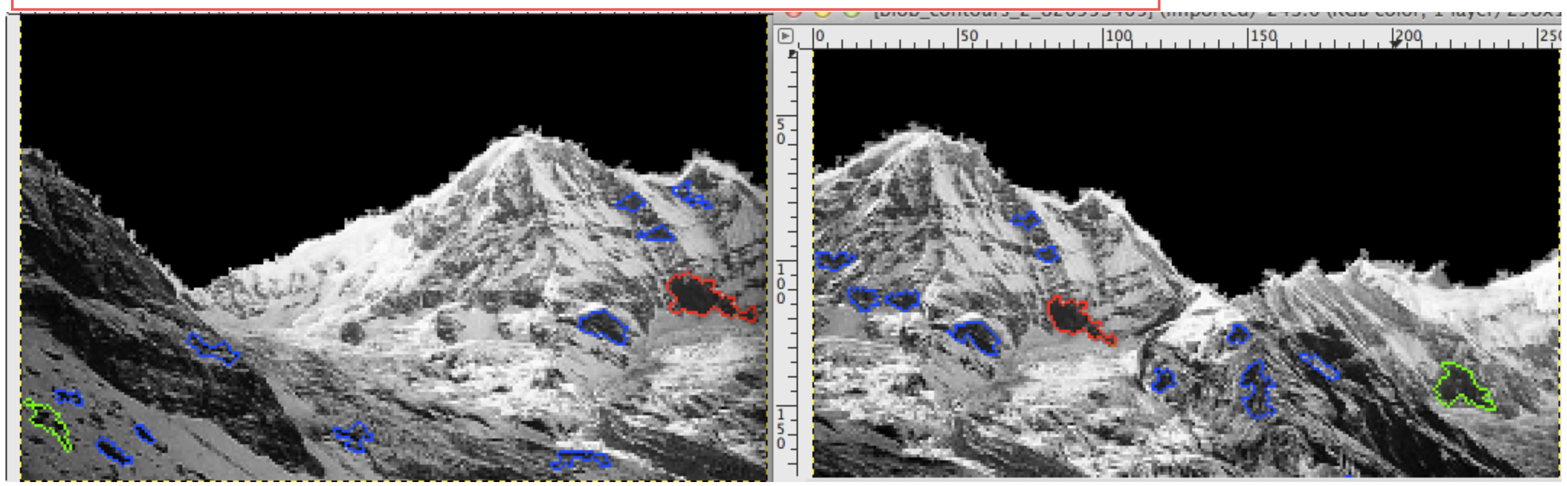




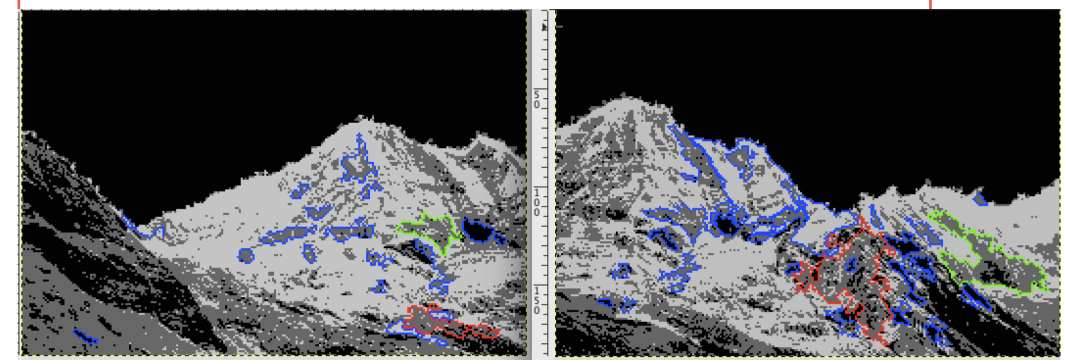


A look at blob contours (to use for scale solutions)

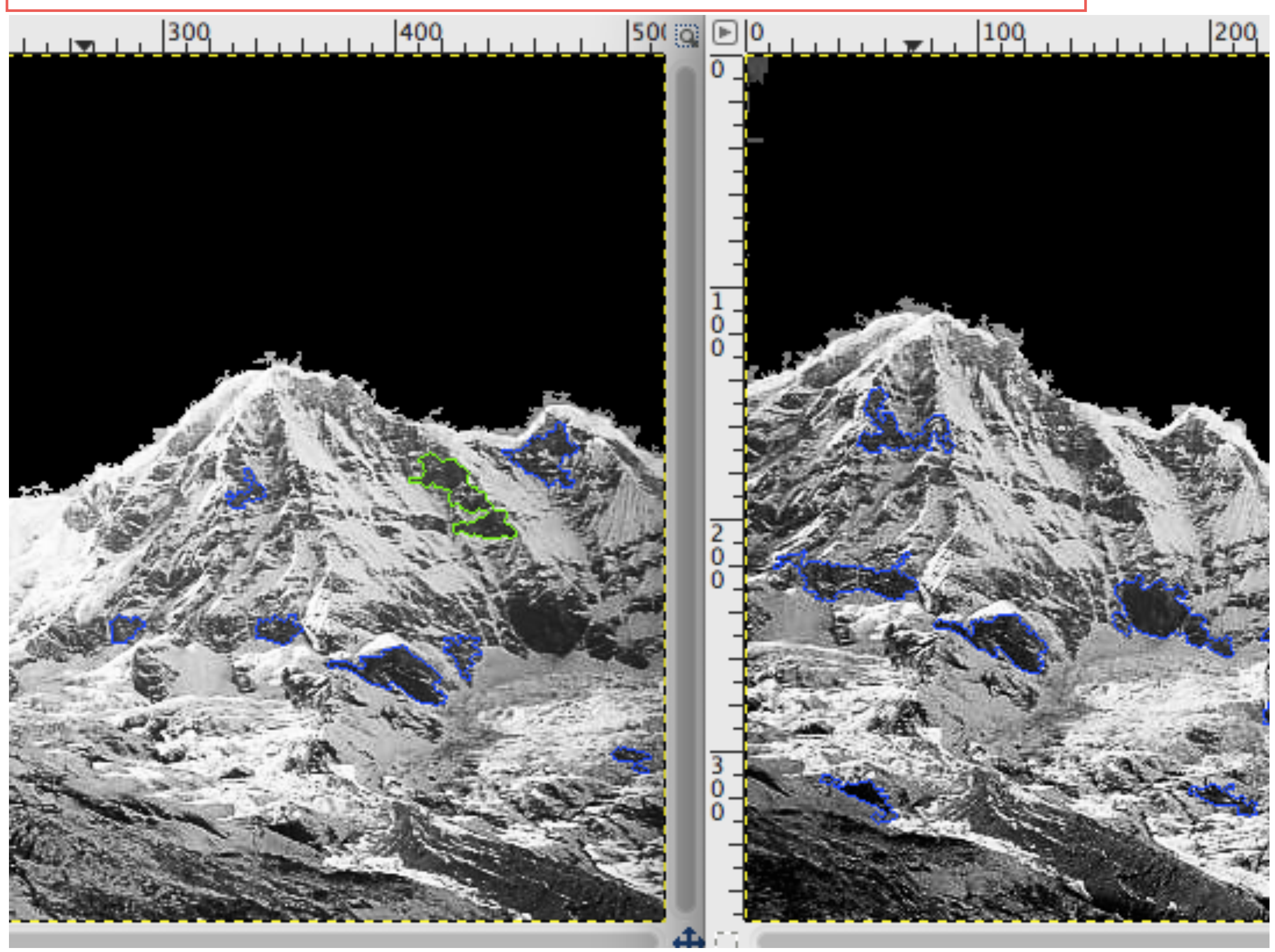
k=2 segmentation blobs, binned to < 300x300



k=3 segmentation blobs, < 300x300



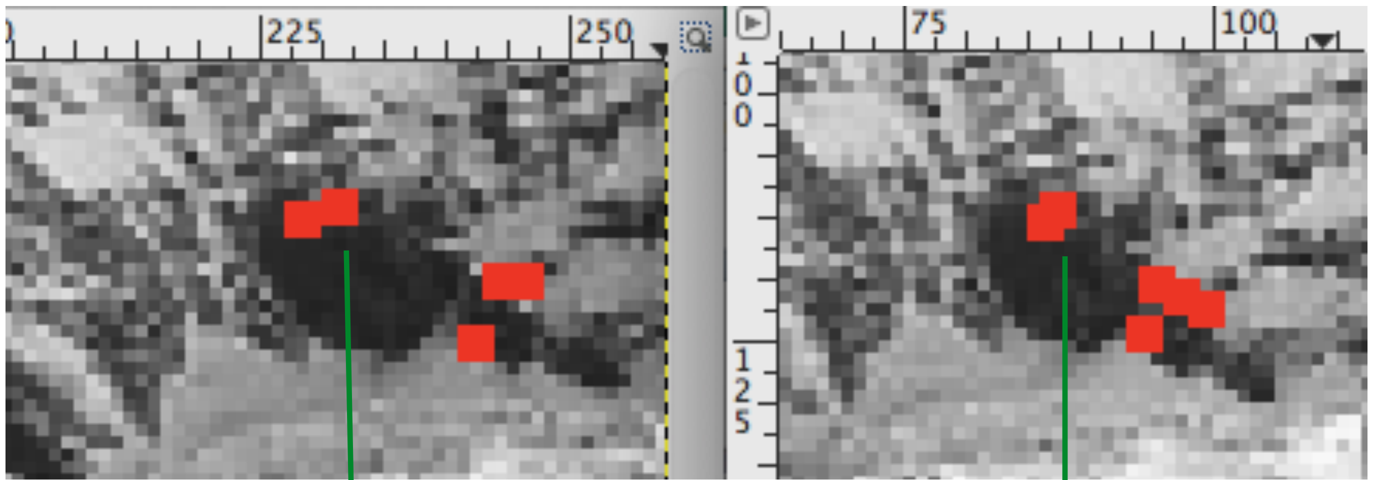
k=2 segmentation blobs, image not binned



In summary with other images, a feasible approach to using blob contours to solve for *scale* may be to use binary segmentation on binned images and images not binned and combine the results w/ expectation that there may only be one or two blobs in the solutions.

k=2 segmentation blobs from images binned to < 300x300, blob perimeters extracted and ordered, perimeter inflection points matched

note, details of the matching contours in a blob are displayed on the binned image without segmentation:




```
[0](236,123) [0](90,120) cost=1.0 scale=0.86  nMatched=4
(245,123) theta1=90    (95,120) theta2=203  intensity=4618.6(4808.2),
(246,123) theta1=45    (97,121) theta2=90
(242,128) theta1=23    (94,124) theta2=225
(228,118) theta1=203   (86,115) theta2=270  intensity=2531.4(2671.8),
(231,117) theta1=135   (87,114) theta2=113  intensity=523.5(1835.1), <====
(244,123) theta1=248   (98,121) theta2=90
```

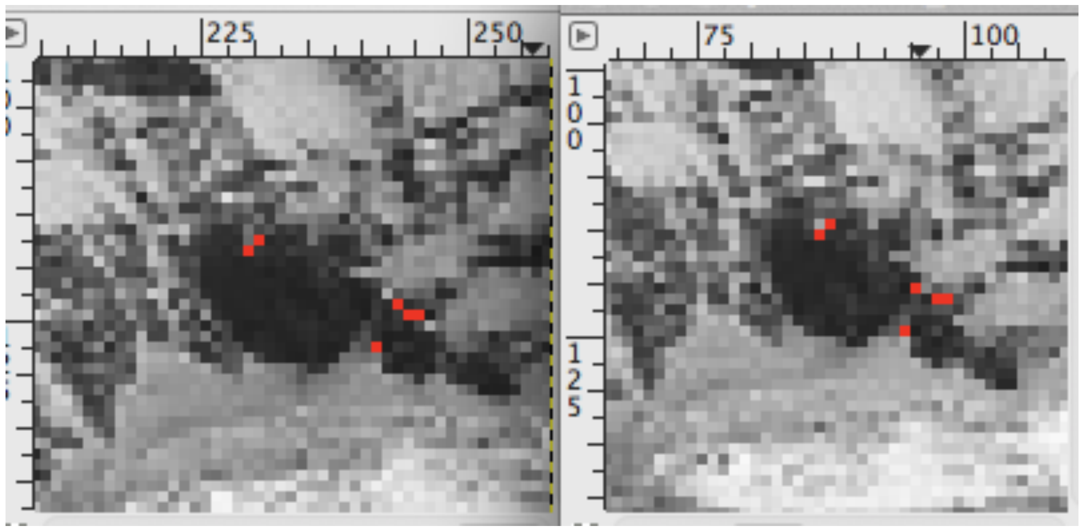
```
A false match to another feature has better overall matches for contour
points so the method has to be improved.  might try best orientation and
increasing dither value.

[0](236,123) [1](225,144) cost=11.0 scale=0.98  nMatched=5
(227,125) theta1=0    (228,142) theta2=203  1048.2(1482.4),
(226,124) theta1=203  (230,144) theta2=248
(245,123) theta1=90   (225,148) theta2=203
(246,123) theta1=45   (225,147) theta2=23
(242,128) theta1=23   (218,140) theta2=293  1223.6(1637.5),
(241,127) theta1=68   (219,139) theta2=338  1414.0(1802.3),
(244,123) theta1=248  (229,143) theta2=225  582.4(2856.6),  <=== best match (but false)
(228,118) theta1=203  (232,148) theta2=248  2557.6(3910.4),
(231,117) theta1=135  (233,148) theta2=270  2662.2(3554.0)
```

to improve, might bootstrap from the best matching point's orientation to correct for errors in blob perimeters due to segmentation, etc..
best orientation applied to other points and a dither of 1 or 2.

one caveat is that the orientation found locally is pointing outwards from the blob, giving a better distinguishing region to compare, but using one orientation for all features instead may sometimes be comparing regions within the blob (the blob was chosen as a relatively uniform patch in images).

Can skip an improved comparison step if the differences between img1 and img2 feature orientations are approximately the same constant number.



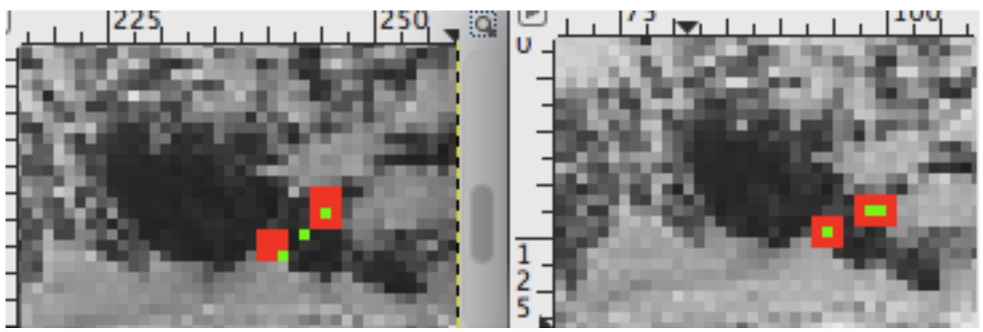

```
[0] [0] dither=2, using previous best rotations for all peaks
(243,123) (95,120) intSqDiff=598.3(2155.0)
(244,124) (97,121) intSqDiff=452.4(2820.2)
(241,127) (94,124) intSqDiff=273.9(4022.0)
(229,118) (86,115) intSqDiff=179.3(1226.4)
(230,117) (87,114) intSqDiff=175.5(1835.1)
(245,124) (98,121) intSqDiff=316.7(2304.6) nMaxStats=6
[0] [1]
(227,123) (228,142) intSqDiff=390.9(1482.4)
(226,123) (230,144) intSqDiff=588.6(1617.9)
(246,124) (225,148) intSqDiff=945.1(2646.4)
(244,124) (225,147) intSqDiff=990.8(2360.6)
(239,129) (219,139) intSqDiff=1345.3(1370.8)
(228,119) (232,148) intSqDiff=1062.1(2164.9)
(230,119) (233,148) intSqDiff=1096.7(1414.1) nMaxStats=9
```

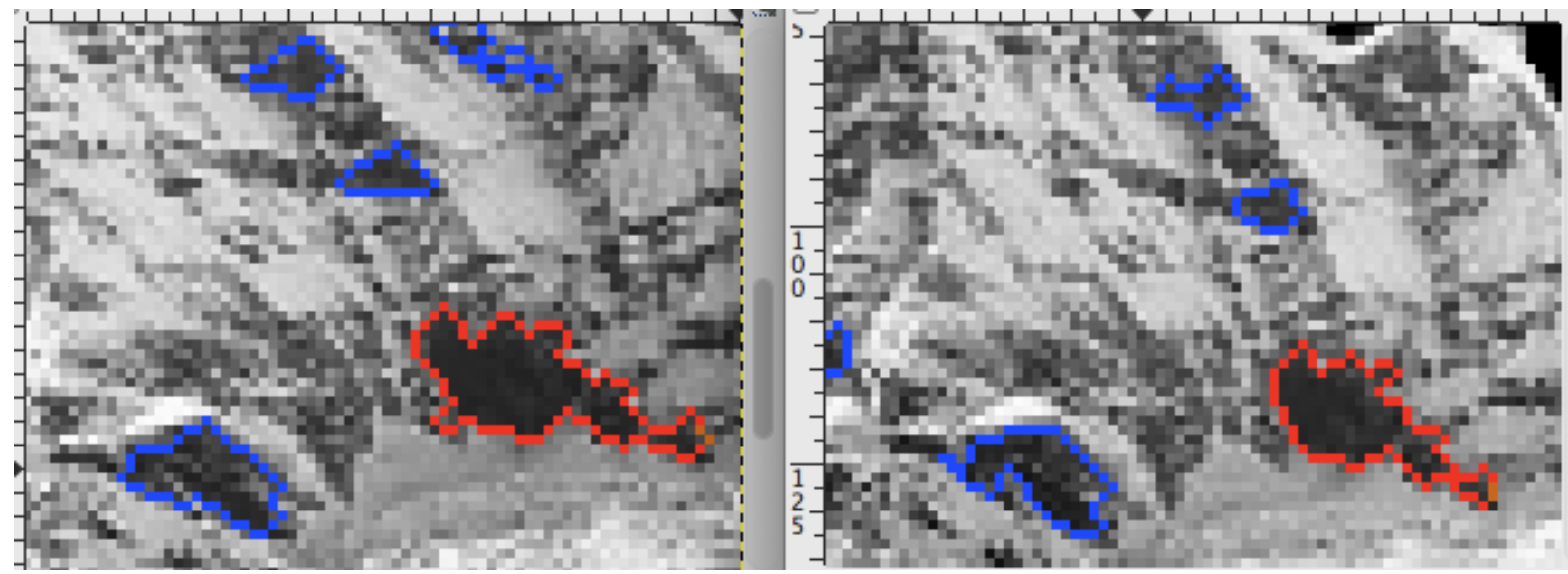
The improvement works pretty well.
The correct match has smaller sum square of differences now making the true match of [0][0] easier to distinguish from the false match of [0][1].

For the **binned images** especially, an edge corrector using the greyscale image as a guide is helpful to remove some of the defects present in the perimeter curve due to blobs extracted as contiguous pixels (in contrast to edges which are extracted from subtracted gaussian blurred images).

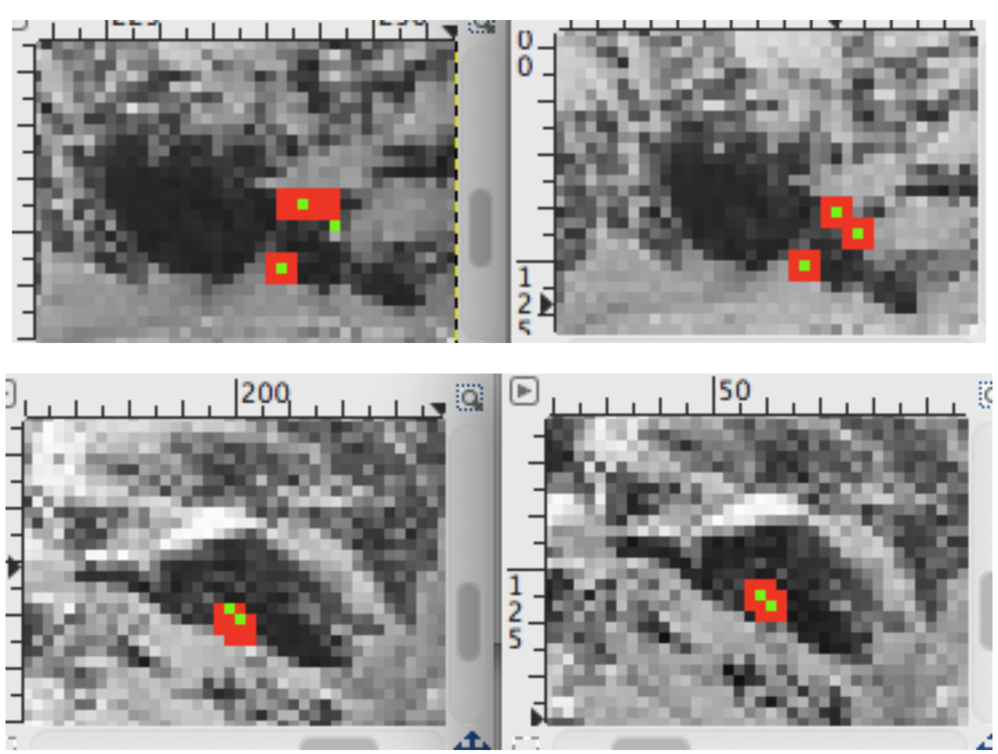
Red are the contour matched points. **Light green** are the feature improved locations.

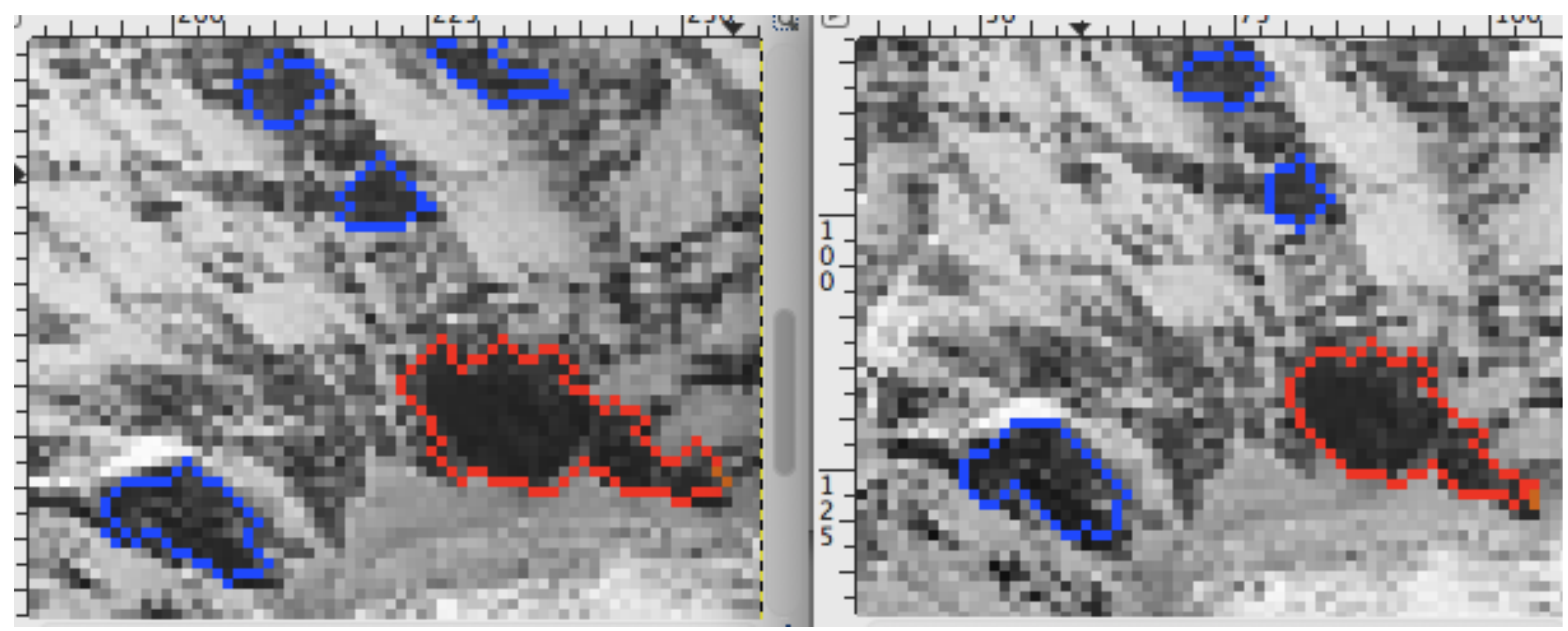
before image guided corrections:





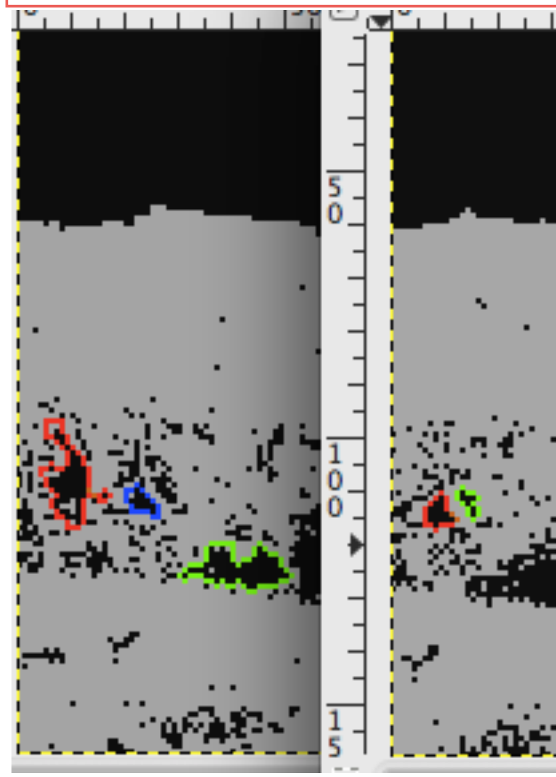
after image guided corrections:



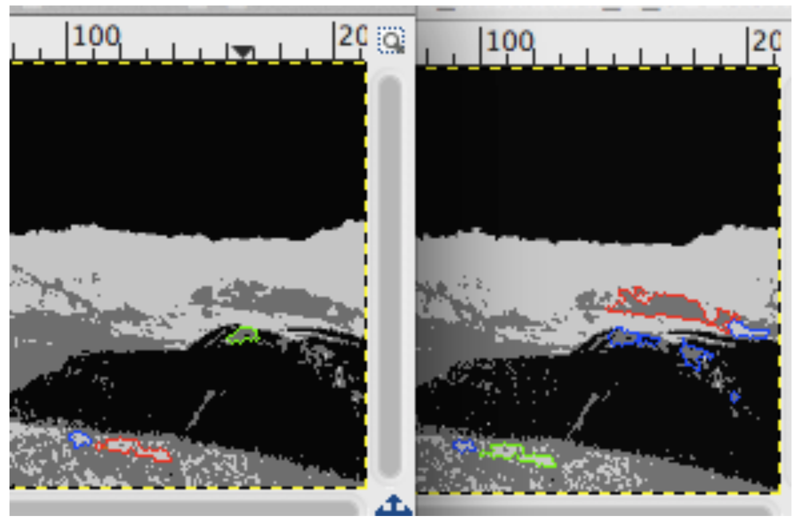


A look at blob contours (to use for scale solutions)

k=2 segmentation blobs, < 300x300, no histeq



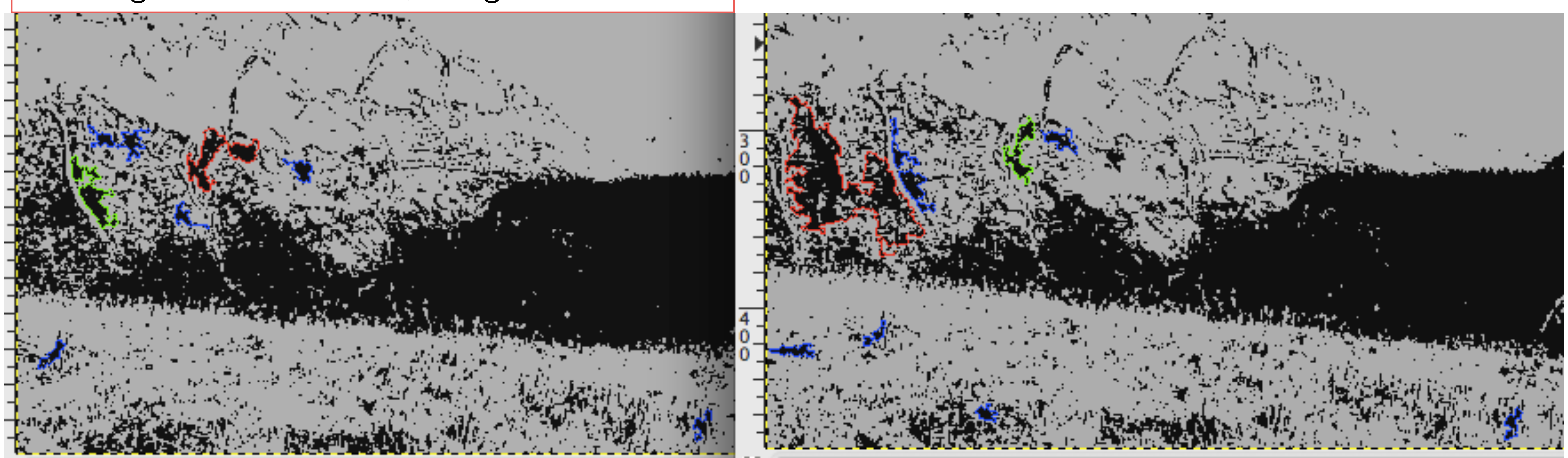
k=3 segmentation blobs, < 300x300, no histeq



In summary with other images, a feasible approach to using blob contours to solve for *scale* may be to use binary segmentation on binned images and images not binned and combine the results w/ expectation that there may only be one or two blobs in the solutions.

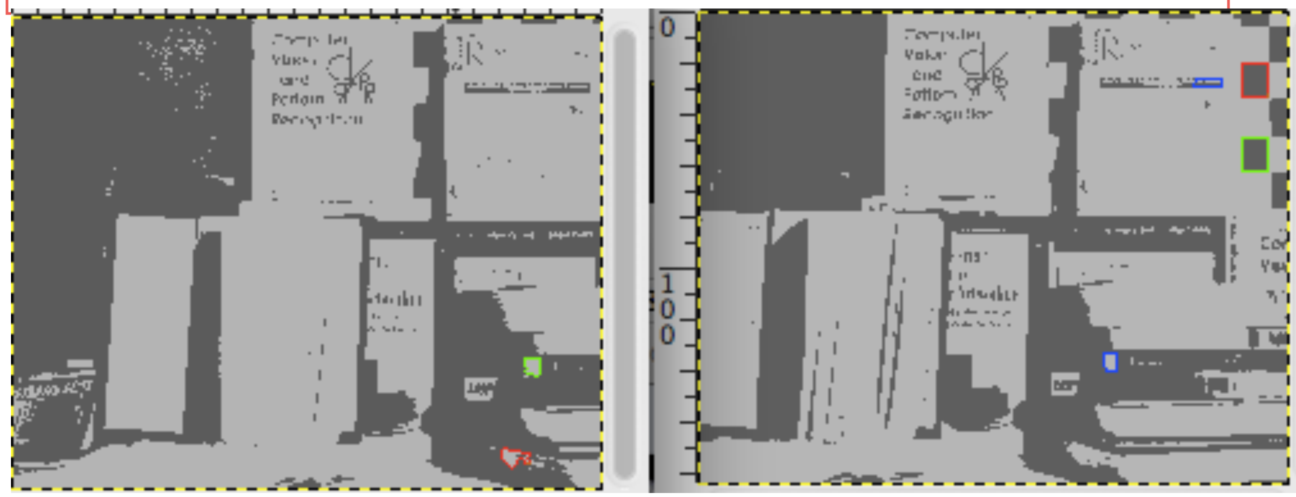
k=3 segmentation blobs, image not binned
has about 4 strong blobs in common

k=2 segmentation blobs, image not binned

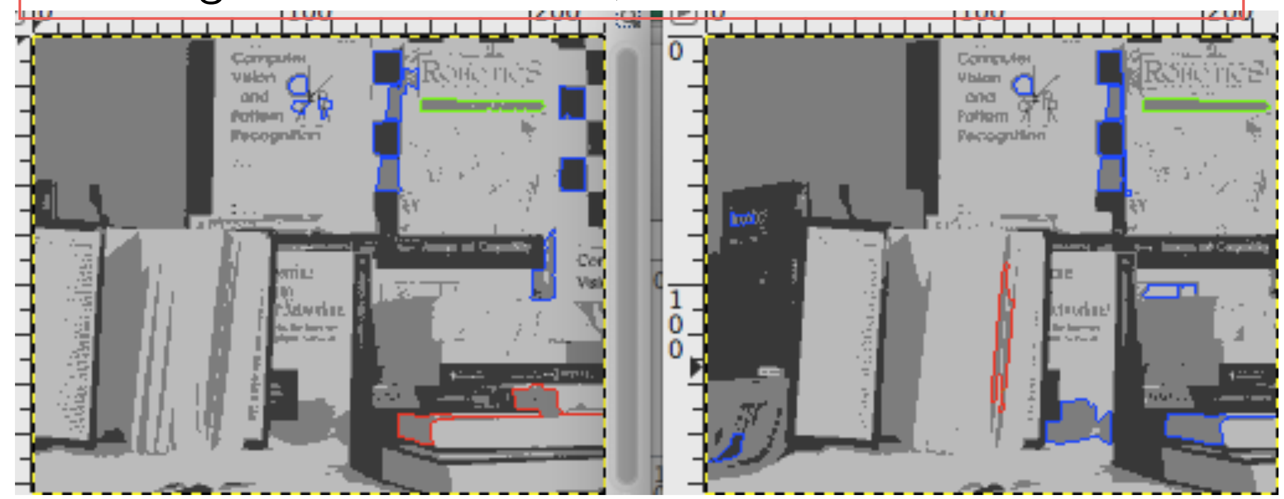


A look at blob contours (to use for scale solutions)

k=2 segmentation blobs, binned to < 300x300

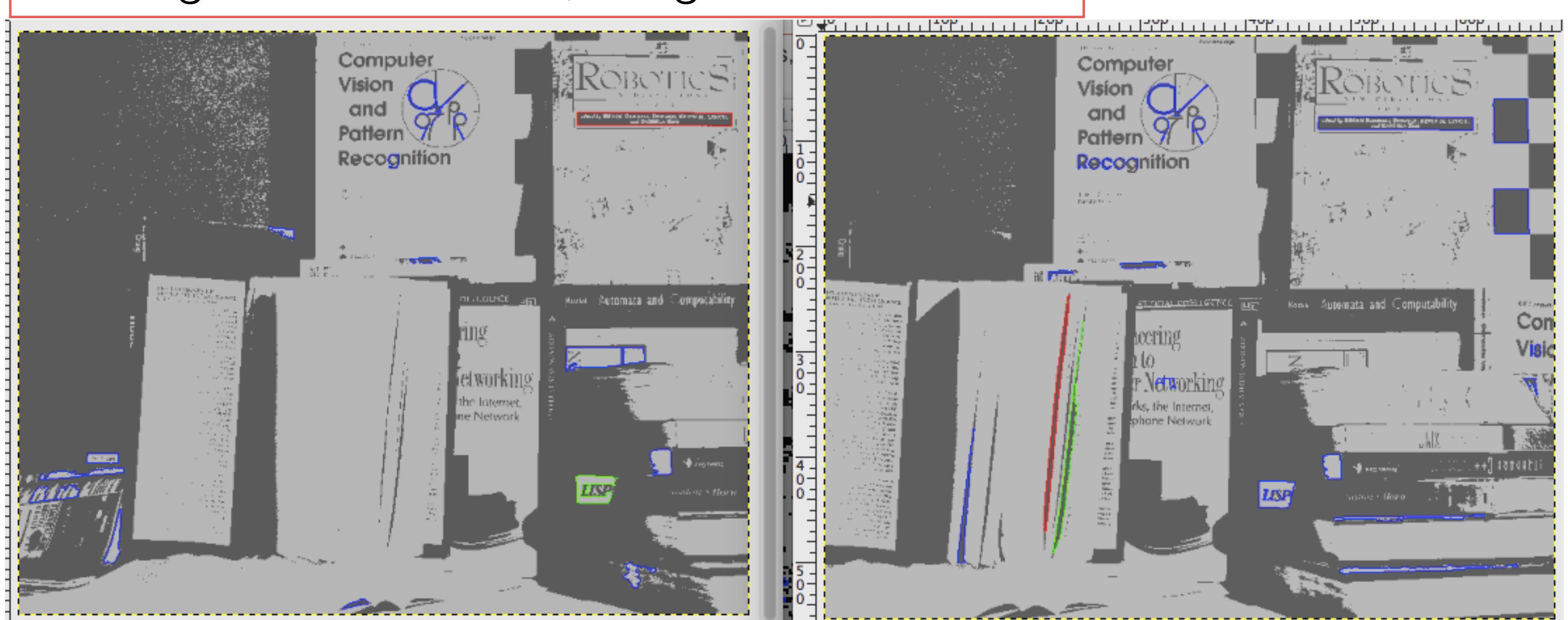


k=3 segmentation blobs, binned to < 300x300

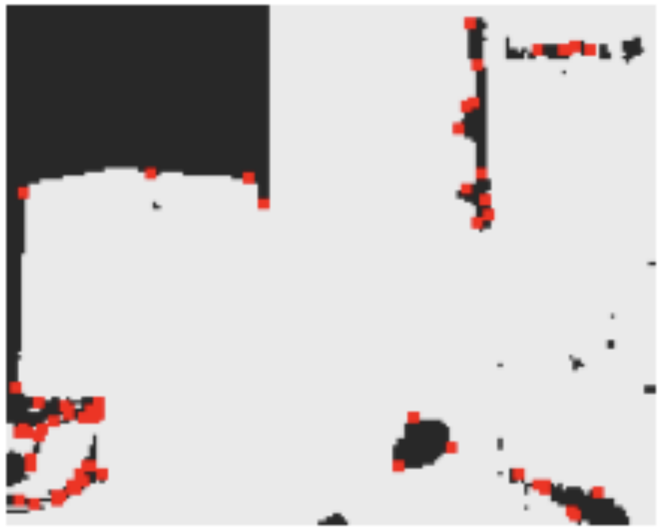


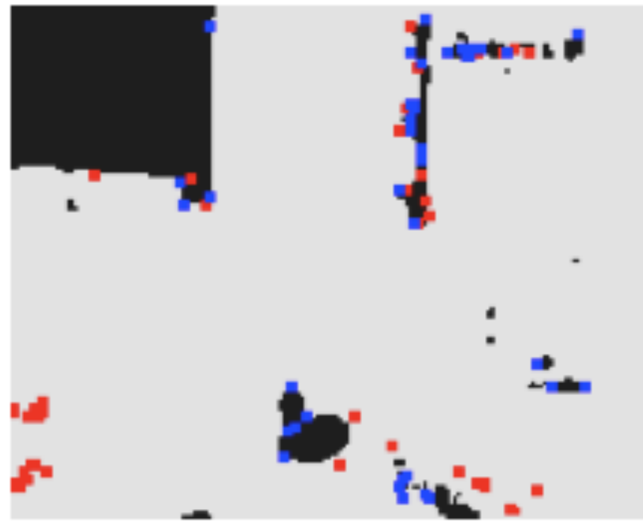
In summary with other images, a feasible approach to using blob contours to solve for *scale* may be to use binary segmentation (k=2) on binned images and images not binned,and combine the results w/ expectation that there may only be one or two blobs in the solutions.
In this case, the number of possible matches for k=2 is very low, but still significant. The k=3 binned to < 300x300, however, has many more features in common.

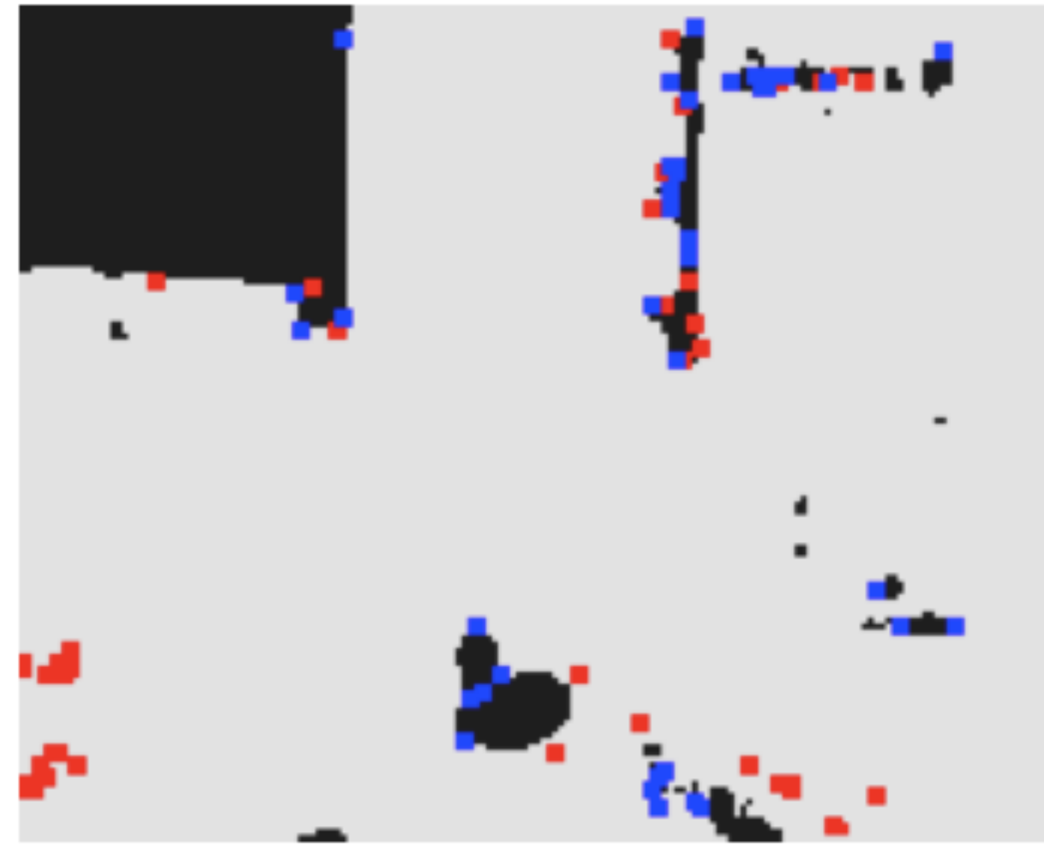
k=2 segmentation blobs, image not binned



A quick look at color segmentation with k=3 followed by binary segmentation, binned to size < 200 x 200, followed by corners worked fine for 2 image sets, but not the third so the other methods in previous slides are preferred (corners filtered to blobs, matching, calculating transformation, then creating correspondence with all corners).







For the stereoscopic above, there's more than translation so this can only be the start of the solution

previous notes from a look at binning and segmentation