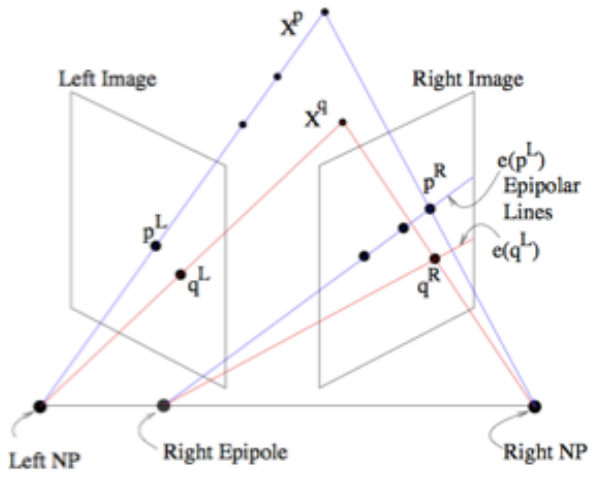
Projection, stereo and panoramic images

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



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.)

NP are the nadir points of the cameras.

The line connecting the left and right NP is the baseline.

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.

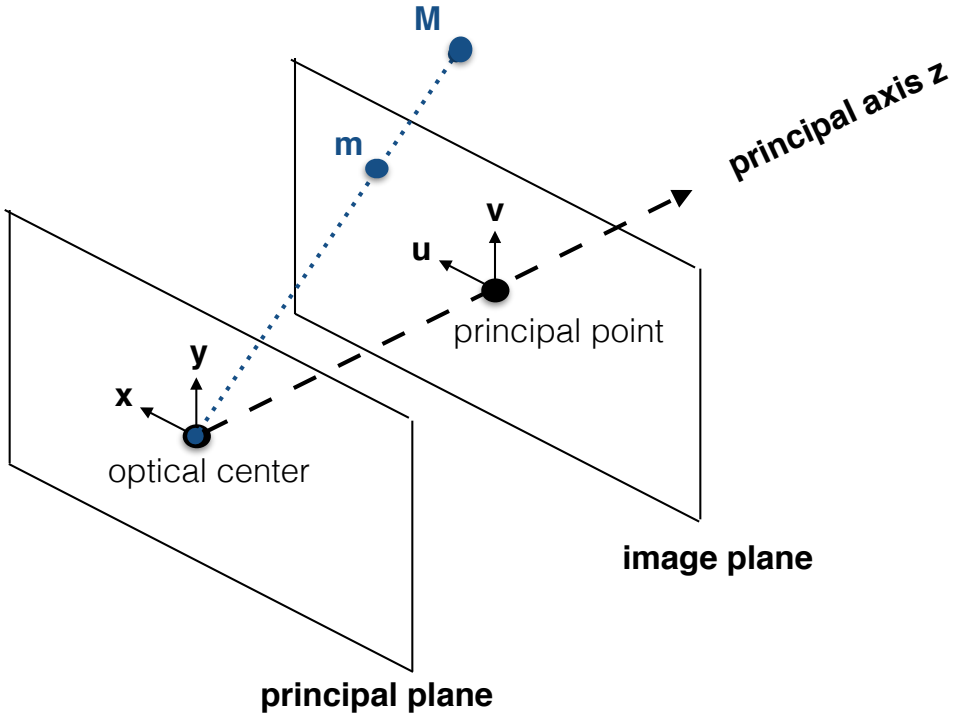
The projection of X^p to left nadir, NP is seen as an epipole line in the right image. If more than one epipole line is present in the right image, they converge at the right epipole (which might not be within the boundaries of the image).

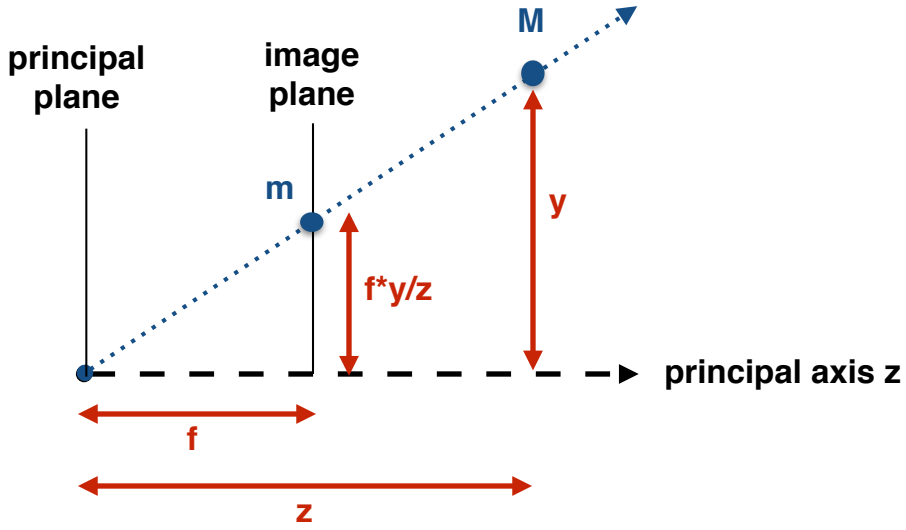
Once the points in the left image, X_L are matched with points in the right image, X_R, if there are at least 7 points, one can determine the "bifocal tensor",
a.k.a. "fundamental matrix, relating the points in the 2 images using a 3x3 matrix of rank 2.

$(X_L)^T * F * X_R = 0$ 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.

Pinhole camera geometry





camera projection matrix P is defined in z***m** = P***M**
where M = $(x,y,z,1)^T$ and m = $(f*x/z, f*y/z, 1)^T$

The right side of the projection eqn can be modified by rotation matrix **R** and translation vector **t** to put the coordinates in the (external) world coordinate system. The matrix is | **R t** |
The six parameters are called *external parameters*. | 0 1 |
rows of **R** and the *optical center* describe the *camera reference frame* in world coordinates.

The left side of the projection eqn can be modified to change coordinates in the image frame.

$$K = \begin{vmatrix} f/s_x & f/s_x*\cot\theta & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{vmatrix}$$

K is the *camera calibration matrix,*
result is pixel coords in image plane.

Pinhole camera geometry (cont.)

intrinsic parameters:
f is the focal distance in mm
o_x, o_y is the principal point in pixel coordinates
s_x is the width of the pixel in mm
s_y is the height of the pixel in mm
ø is the angle between the axes, usually 90 degrees
The aspect ratio s_x/s_y is usually 1

P can be rewritten as P = **K***[I | 0]***G** = **K***[**R** | **t**]

A scale factor **ƛ** applied is P = **ƛ*****K***[**R** | **t**]
This is a 3x4 full rank matrix and can be factorized by QR factorization.

P can be rewritten as P = [$\mathbf{P_{3x3}}$ | $\mathbf{p_4}$] where $\mathbf{P_{3x3}}$ is the first 3 rows of P and $\mathbf{p_4}$ is the 4th column.

If **ƛ**=1, the matrix is normalized and the distance of **M** from the focal plane of the camera is *depth*.

For the image plane at infinity, the image of points doesn't depend on camera position.
The angle between 2 rays is cos θ = $\mathbf{m1^T * \omega * m2 / (}$sqrt$\mathbf{(m1^T * \omega * m1) * (}$sqrt$\mathbf{(m2^T * \omega * m2)}$

(Notes on pinhole camera followed by a few notes on multi-view geometry are from http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO4/tutorial.html)

multi-view geometry

With point correspondences $\mathbf{m_i} \longleftrightarrow \mathbf{M_i}$, , can solve for camera projection matrix P. Using the Kroenecker delta product and vec operator, coefficients are rewritten to their linearly independent forms of a matrix of size $2n \times 12$ called the coefficient matrix A

From a set of n point correspondences, we obtain a $2n \times 12$ coefficient matrix A, where n must be > 6.

In general A will have rank 11 (provided that the points are not all coplanar) and the solution is the *1-dimensional right null-space of A*.

The linear system of equations for inexact data is solved using least squares.
The least-squares solution for vec P^T is the singular vector corresponding to the smallest *singular value* decomposition of A and the direct linear transform.
The equation is usually written in form A * h = 0 where h is vec P^T (its the one dimensional reshaping of the homograph matrix).

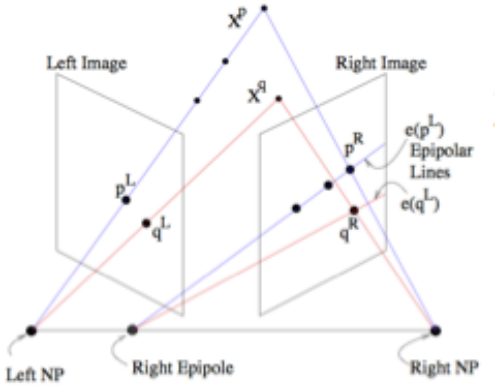
multi-view geometry (cont.)

In general:
— dot product of a point and a line is zero if the point lies on the line
— if the *intrinsic parameters* are known, the relationship between corresponding points is given by the *essential matrix*: $\mathbf{m_r^T * E * m_l^T}$
— when no camera details are available, one can use the *fundamental matrix*.
It's a 3x3 rank 2 homogeneous matrix. It has 7 degrees of freedom.
For any point $\mathbf{m_l}$ in the left image, the corresponding epipolar line $\mathbf{l_r}$ in the right image can be expressed as $\mathbf{l_r = F * m_l}$ and vice versa $\mathbf{l_r = F^T * m_r}$
— the *essential and fundamental matrices* are related through $\mathbf{F} = \mathbf{K_r^{-T}} * \mathbf{E} * \mathbf{K_l^{-T}}$ where Kr and Kl are the left and right camera matrices

Regarding 3D reconstruction:
— when both *intrinsic* and *extrinsic* camera parameters are known, the reconstruction is solved unambiguously by triangulation.
— when only *intrinsic* parameters are known, extrinsic parameters can be estimated and the reconstruction can be solved up to an unknown scale factor, that is **R** can be estimated, but **t** can be only up to a scale factor. the epipolar geometry is the essential matrix and the solvable projection is euclidean (rigid+ uniform scale)
— when neither *intrinsic* nor *extrinsic* parameters are known, i.e., the only information available are pixel correspondences, the reconstruction can be solved up to an unknown, global projective transformation of the world.

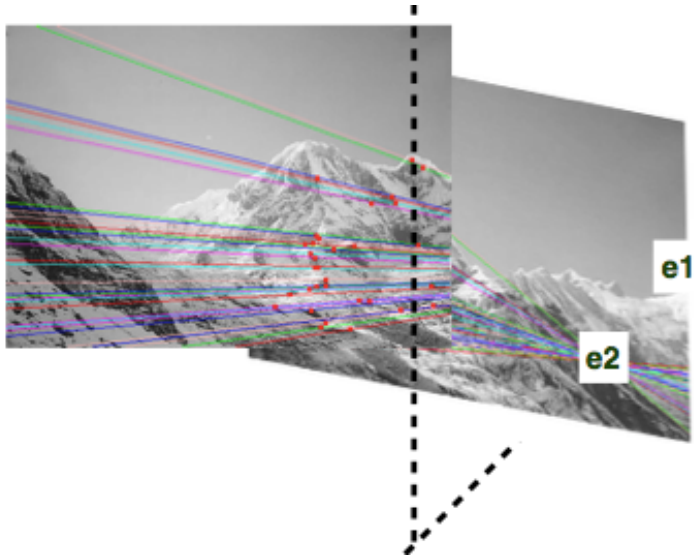
In Trifocal geometry, a trifocal tensor is used.

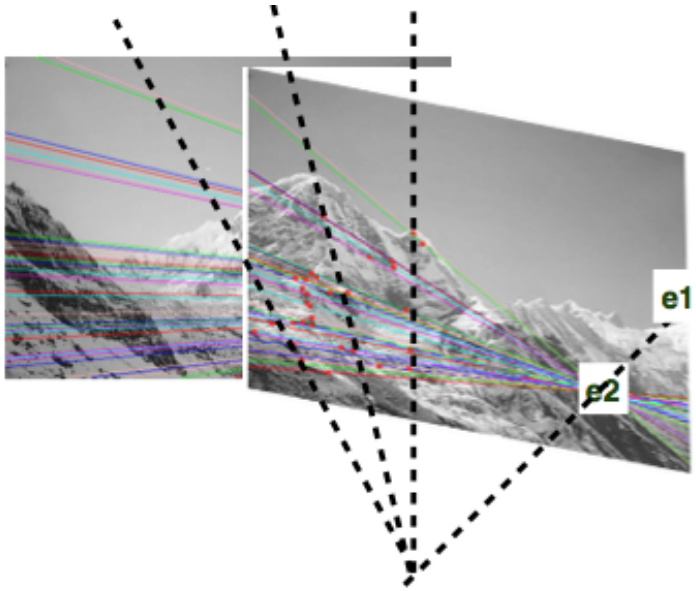
Projection, stereo and panoramic images



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



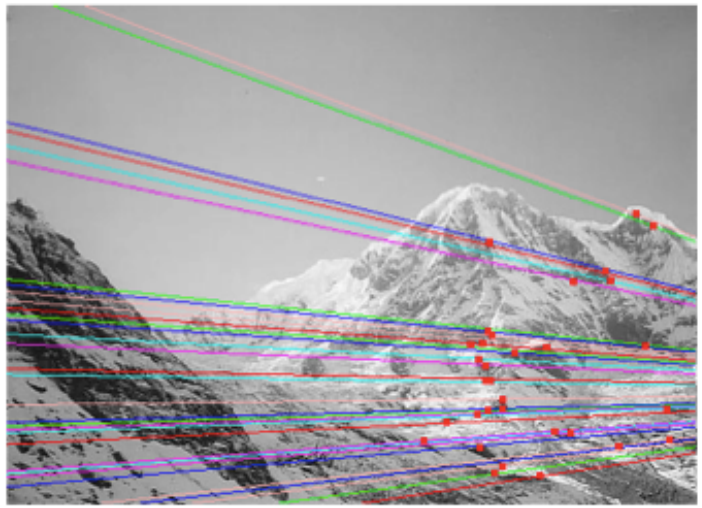


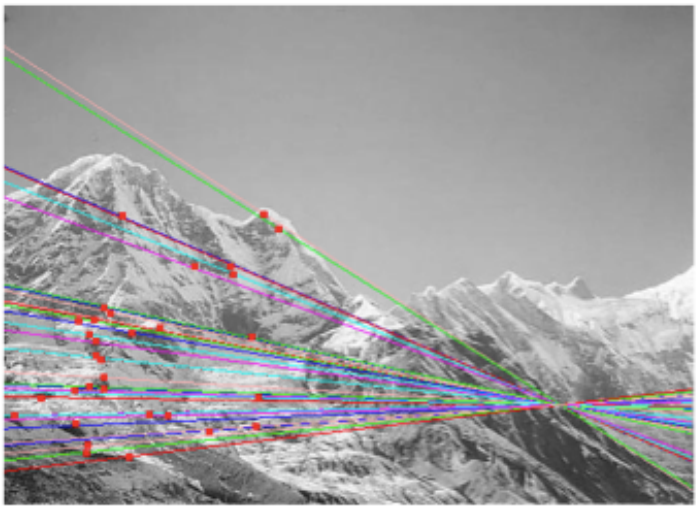
same camera objectives (nadirs), but different orientation for the 2 images (that is, rotated around the same nadir)

Point Correspondence

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

manually making a point list from the corners from the edge extractor used with "outdoor mode":

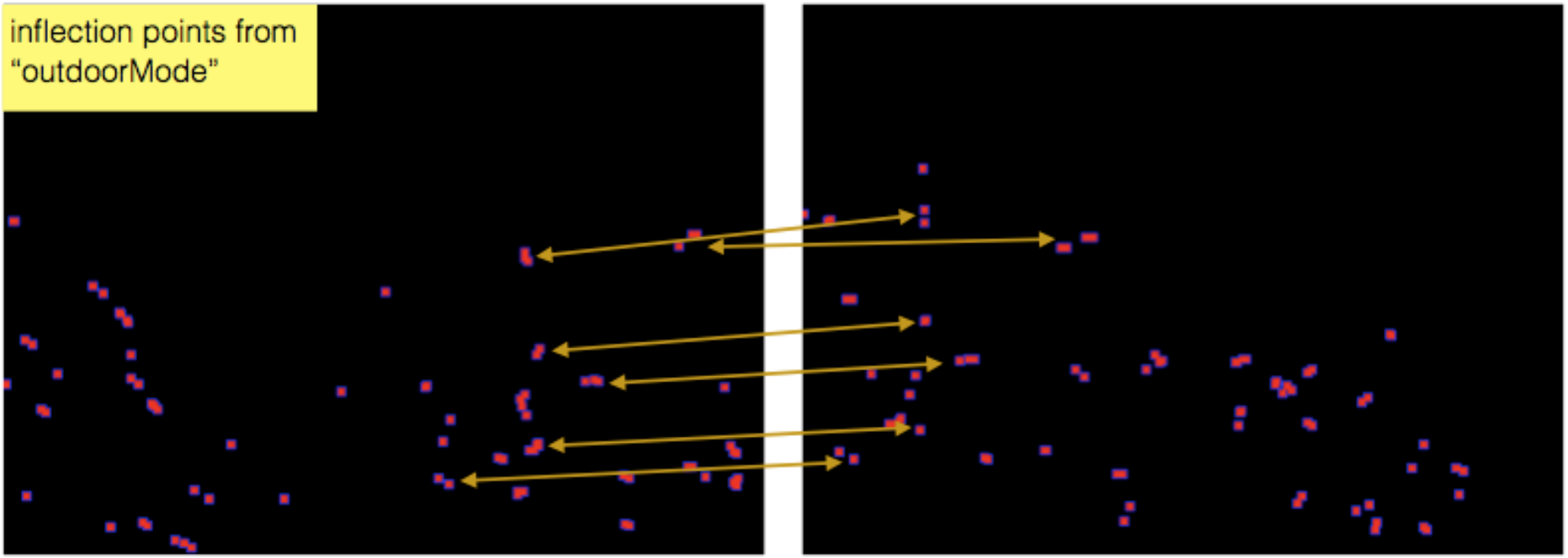




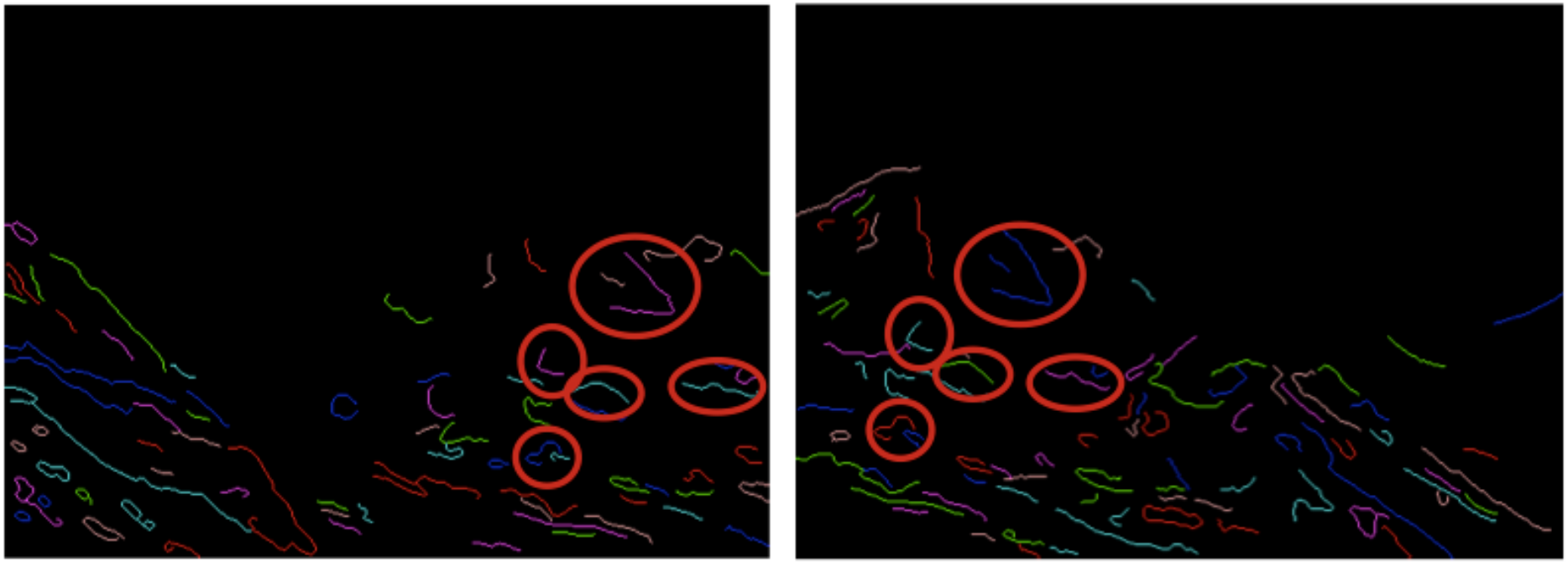
stereo projection fit to 32 points already known to match shows what the epipolar projections should be when the corner find + corner match + stereo projection solve are correctly automated.

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

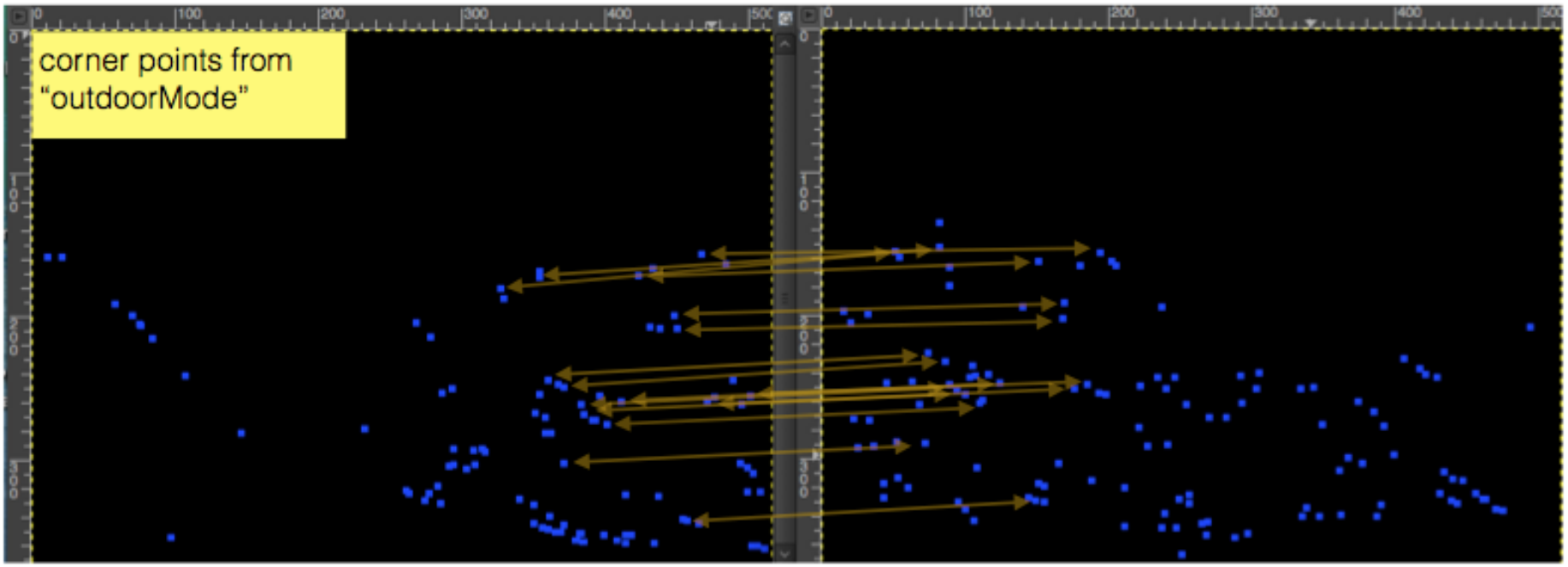
The Brown & Lowe 2003 images: point matching difficult because image intersection << difference



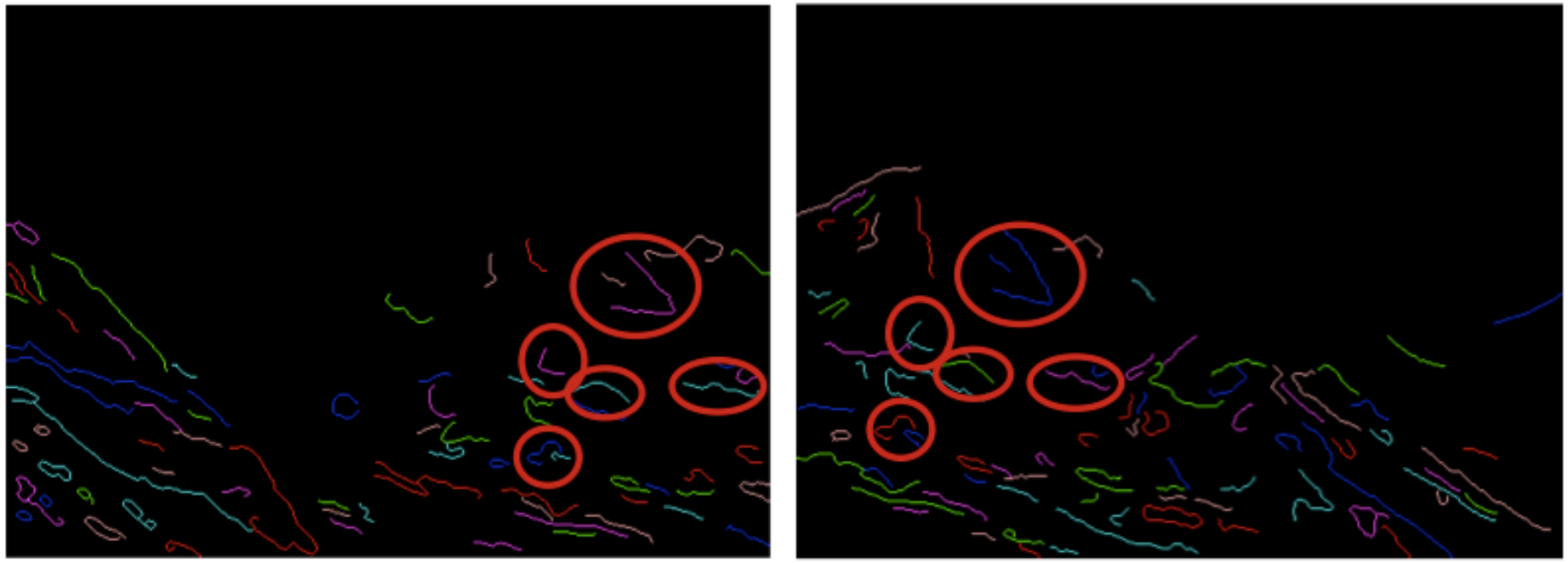
could consider distinct curves and their immediate neighbors, but that would be many more points:



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



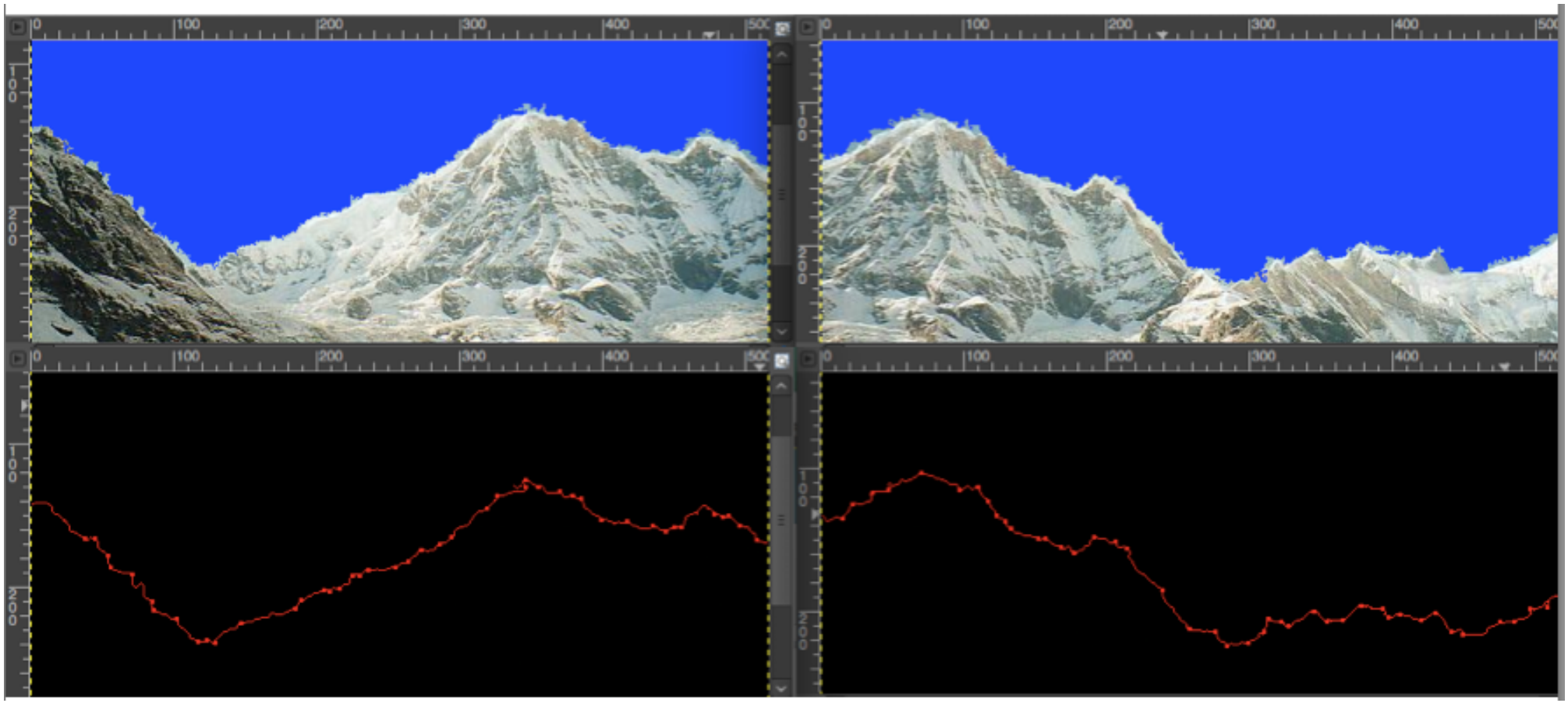
could consider distinct curves and their immediate neighbors, but that would be many more points:



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)

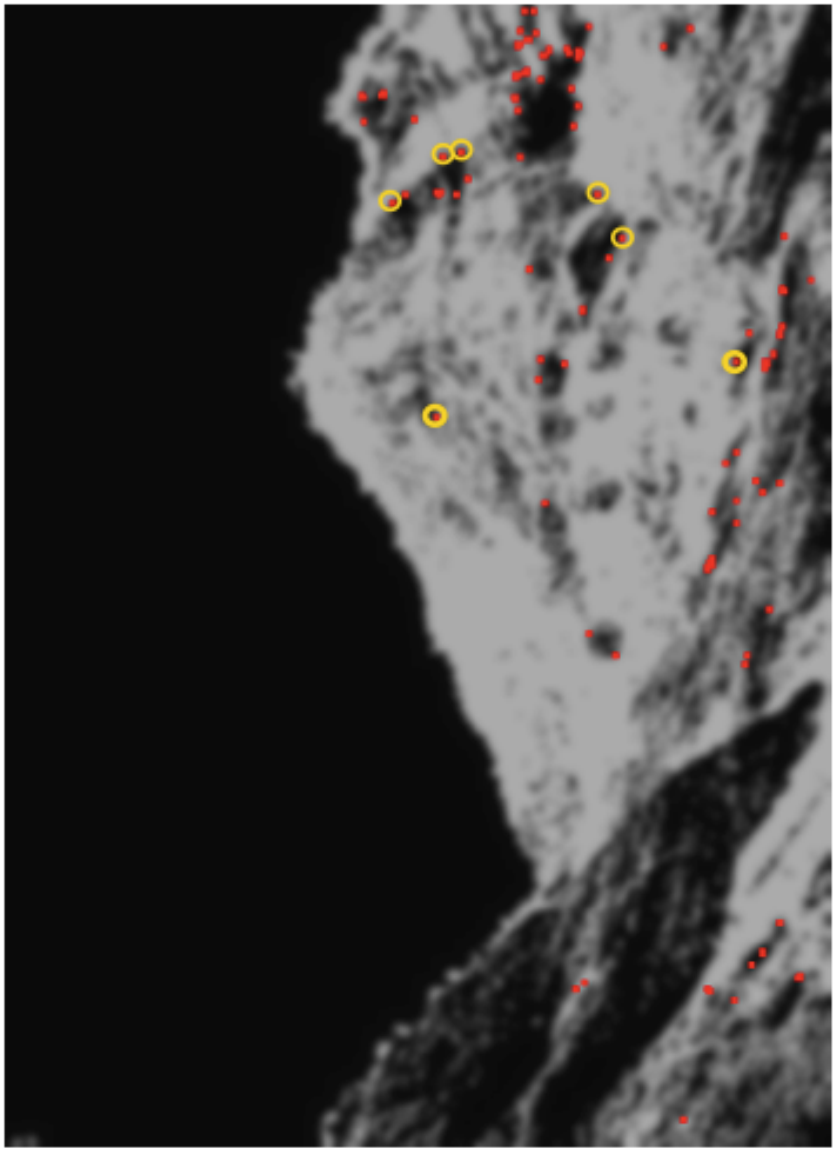
This sky mask can help to pre-process the image before feature finding and correspondence.

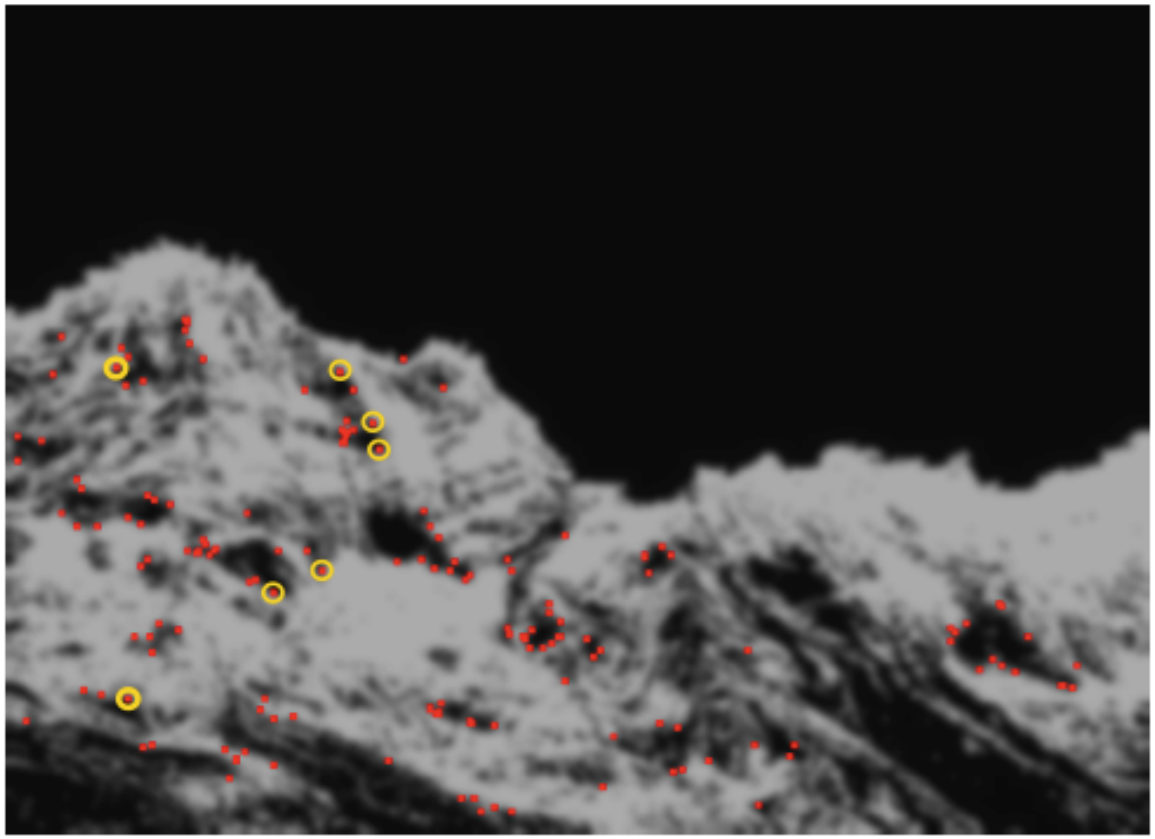
Note that the skyline corners will be improved before real use, this is a quick look.



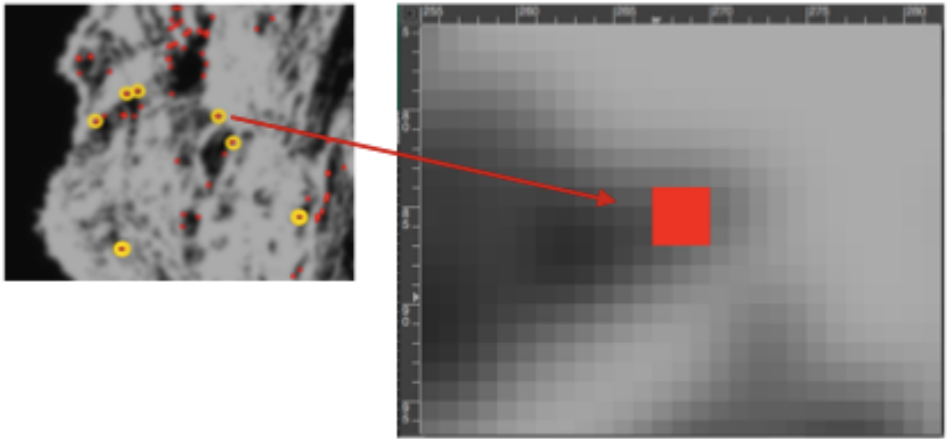
Need to match the corners using **features** because corner locations alone are often not enough to match correctly, especially when the number of truly matchable is smaller than the number of corners or when objects such as ridges exist giving large number of false matches to a different ridge of corners.

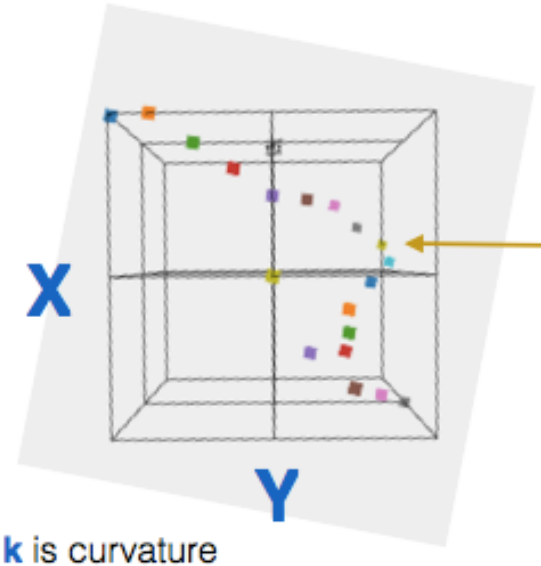
For features, need to determine an orientation of the corner region which is consistent w.r.t. the corner in any image. The highlighted corners are some examined in detail. The left is rotated to make sure the methods correctly handle rotation.



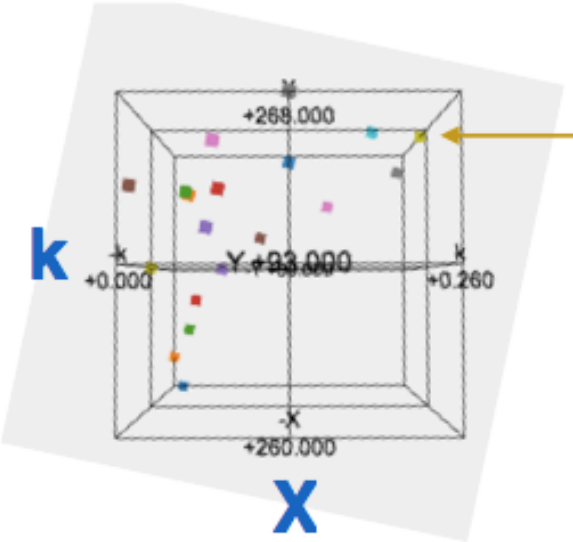


orientation of the feature: the edge already exists from making the corner, so can determine the tangent at the corner.





k is curvature

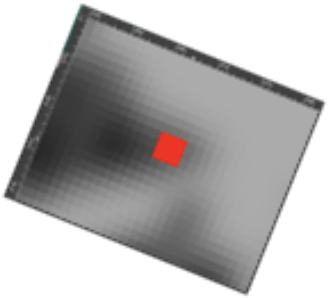


maximum in curvature

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).

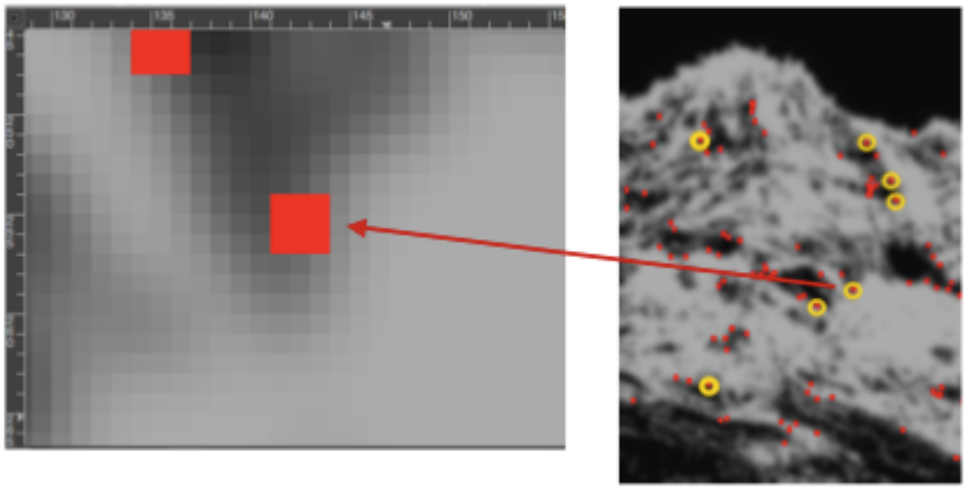
```
deriving the perpendicular angle from the points directly to
the left and right of the maximum of curvature for points which
have curvature
k        x     y
0.24, 267, 84          dx,dy=(1,1), perp=45
0.26, 268, 85 <-max k
0.21, 268, 86          dx,dy=(0,1), perp==0

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

rotated



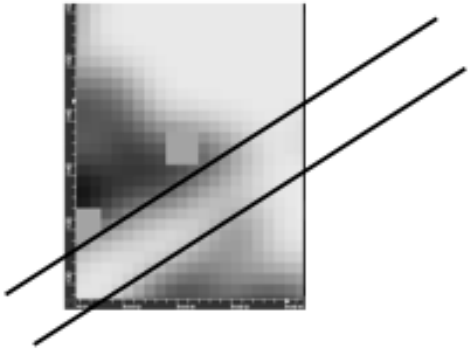
```
extract curvature
from the highest
resolution
scale space curves:

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.



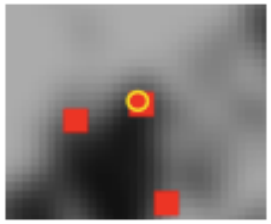
```
extract the final curvature from the highest resolution
scale space curves:
idx=112 (143.0,255.0)

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

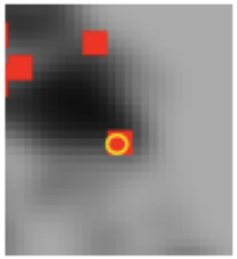


rotated

orientation. corners from Brown & Lowe 2003 left and right panorama images



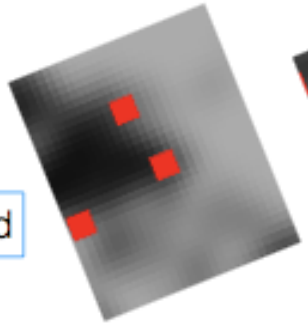
k[0]=0.25 x,y=(204, 66)
k[1]=0.32 x,y=(205, 66)
k[2]=0.32 x,y=(206, 66)
k[3]=0.30 x,y=(207, 67)
k[4]=0.26 x,y=(208, 68)



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)

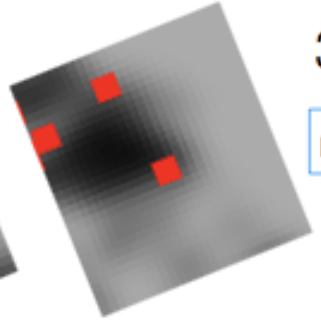
67.5

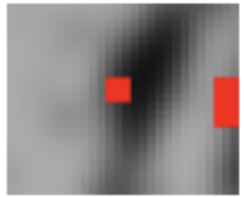
rotated



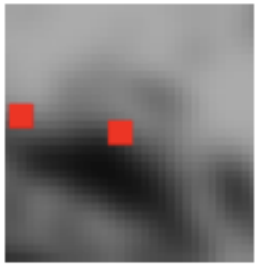
337.5

rotated





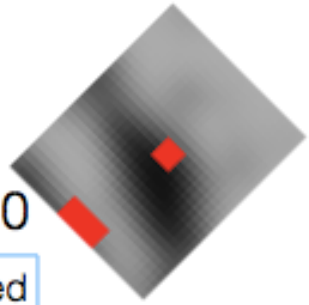
k[0]=0.11 x,y=(332, 161)
k[1]=0.37 x,y=(331, 161)
k[2]=0.58 x,y=(330, 161)
k[3]=0.47 x,y=(330, 162)
k[4]=0.30 x,y=(330, 163)

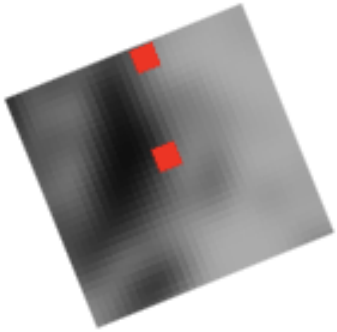


k[0]=0.07 x,y=(56, 315)
k[1]=0.09 x,y=(55, 314)
k[2]=0.31 x,y=(54, 313)
k[3]=0.23 x,y=(53, 313)
k[4]=0.03 x,y=(52, 313)

135.0

rotated

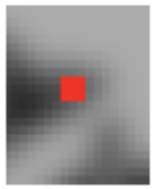




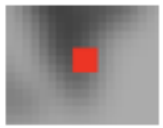
67.5

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)

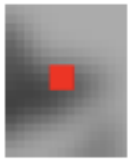


k[0]=0.13 x,y=(143, 254)
k[1]=0.34 x,y=(143, 255)
k[2]=0.43 x,y=(142, 255)
k[3]=0.35 x,y=(141, 255)
k[4]=0.28 x,y=(140, 255)



22.5

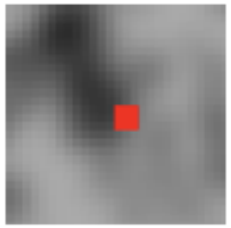
rotated



270.0

rotated

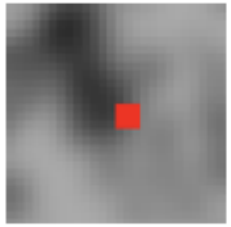
k[0]=0.16 x,y=(195, 183)
k[1]=0.26 x,y=(195, 184)
k[2]=0.32 x,y=(195, 185)
k[3]=0.29 x,y=(195, 186)
k[4]=0.25 x,y=(194, 187)



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

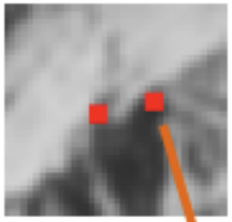
rotated

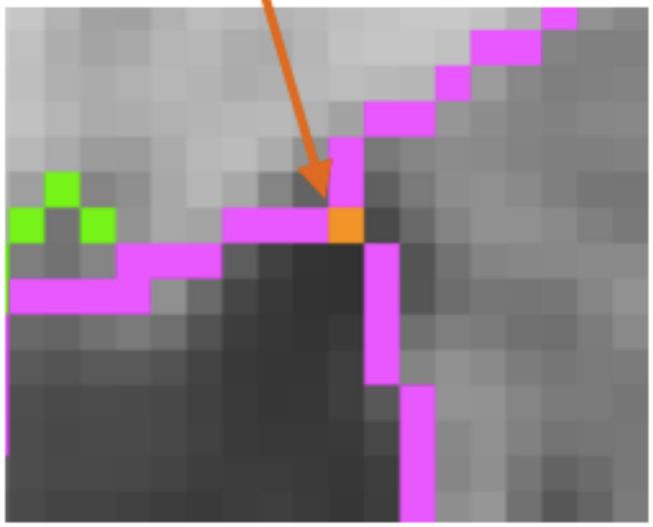
orientation and junctions. corners

from Brown & Lowe 2003 left and right panorama images

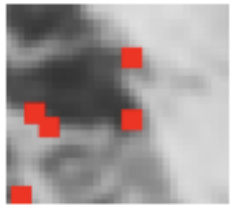
```
k[0]=0.09  x,y=(204, 65)
k[1]=0.19  x,y=(205, 65)
k[2]=0.21  x,y=(206, 65)
k[3]=0.09  x,y=(206, 64)
k[4]=0.03  x,y=(206, 63)
```

315.0



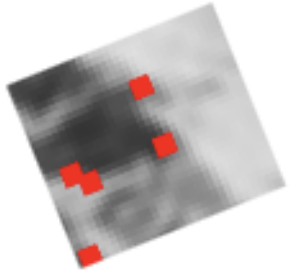


```
k[0]=0.21  x,y=(170, 198)
k[1]=0.32  x,y=(170, 199)
k[2]=0.41  x,y=(170, 200)
k[3]=0.38  x,y=(169, 201)
k[4]=0.27  x,y=(168, 201)
```



337.5

rotated

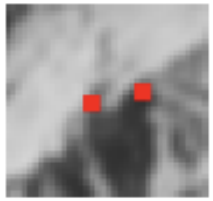


The left image corner is in a *junction* and that requires more complex analysis to determine here that the better edge for it is the lower right edge instead of the upper edge (the later gives an inconsistent orientation w.r.t. the right image).

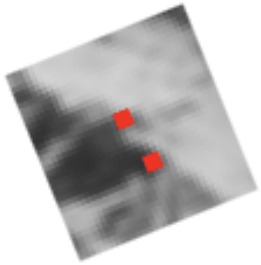
Without the more complex analysis for now, will just create a CornerRegion for each combination and one of the 3 will match the right.

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)
```

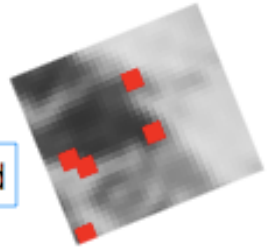


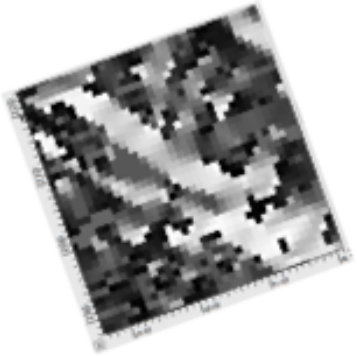
67.5

rotated

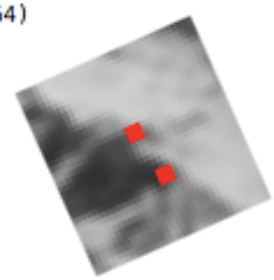
337.5

rotated



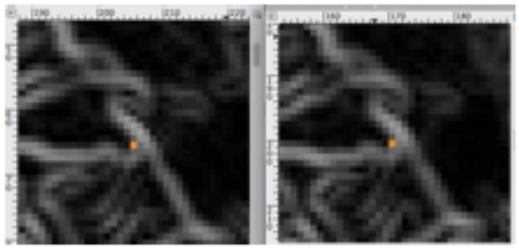


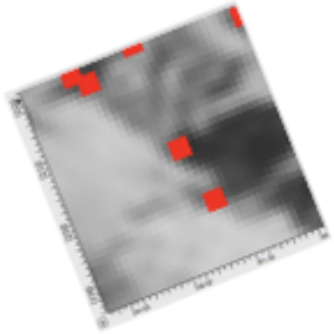
```
k[0]=0.02  x,y=(205, 65)
k[1]=0.21  x,y=(206, 65)
k[2]=0.02  x,y=(206, 64)
```



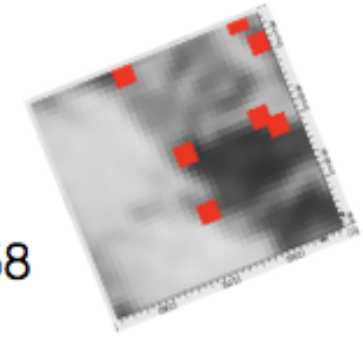
66

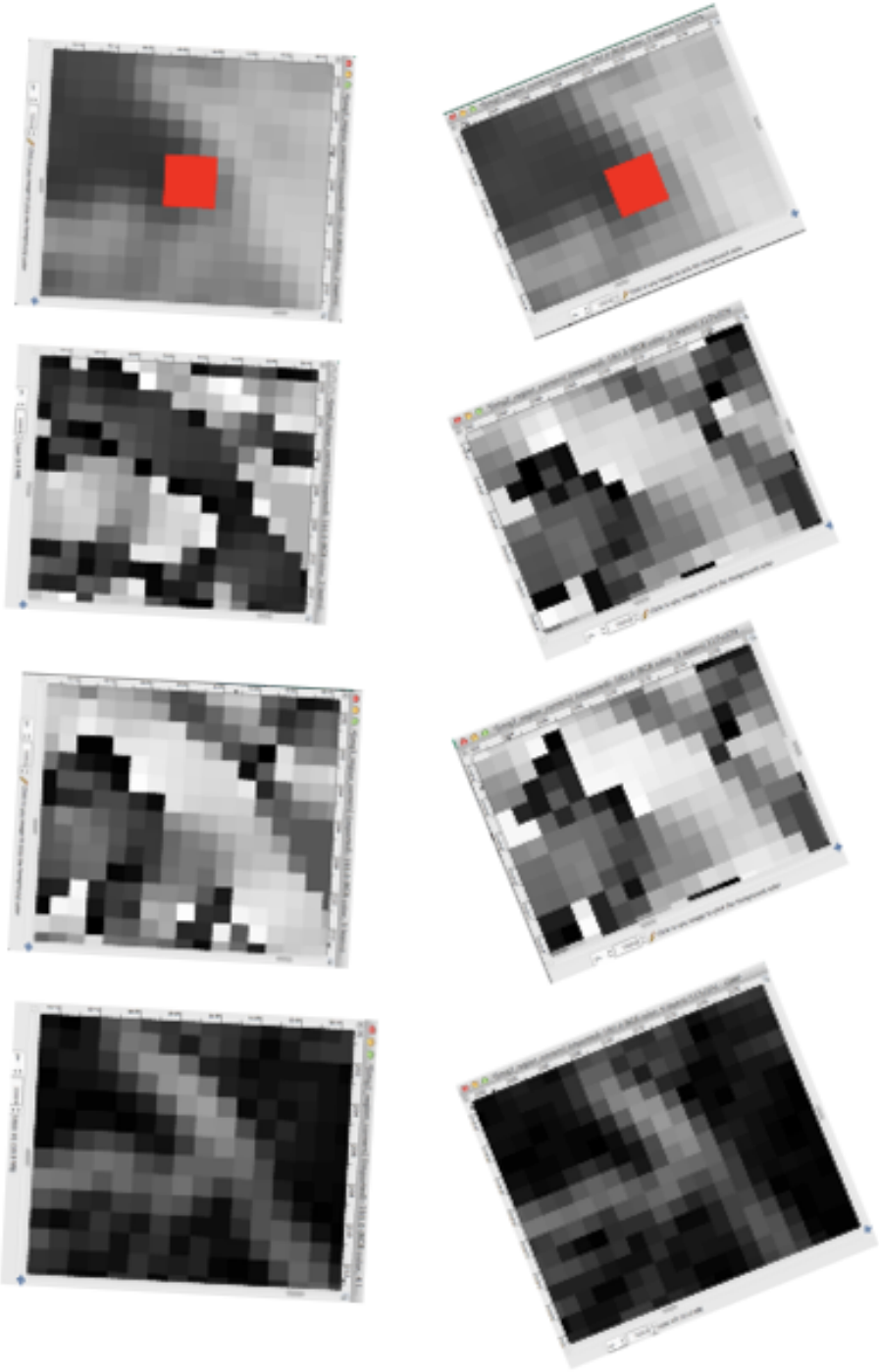
rotated





250 158





93

338

greyscale rotated to “dominant orientation”

gradient theta rotated to “dominant orientation”

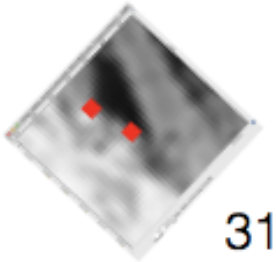
gradient theta rotated to “dominant orientation” with “dominant orientation” subtracted from theta

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

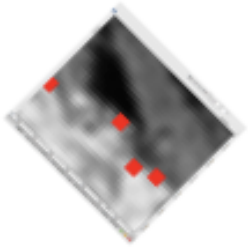
can see that SSD on gradient theta could be a good descriptor with the orientation corrections and quadrant consideration (e.g. 0 diff 350 is 10, not 350)

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.

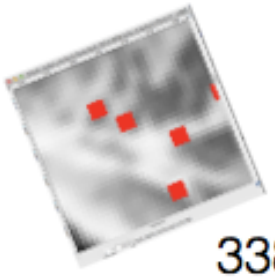


313

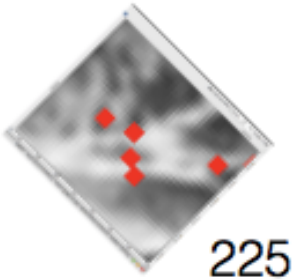


225

[1]

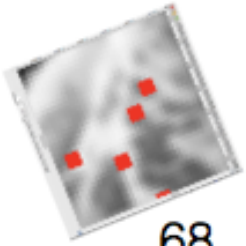


338



225

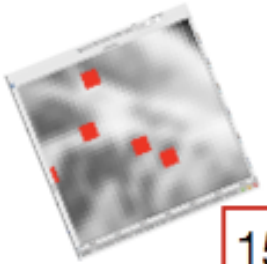
[2]



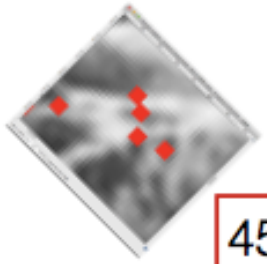
68

293

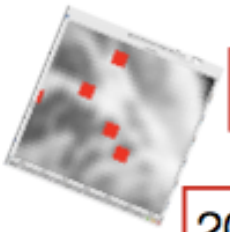
gradient theta rotated to "dominant orientation" with "dominant orientation" subtracted from theta (ref for method?)



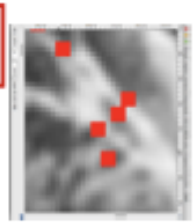
158



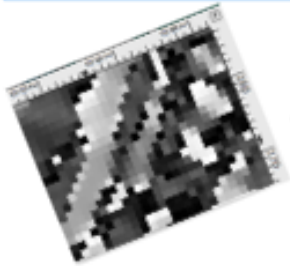
45



90



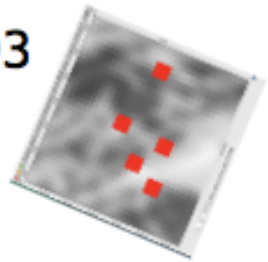
202

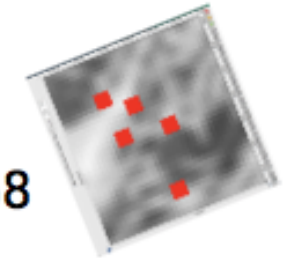


68

315

293





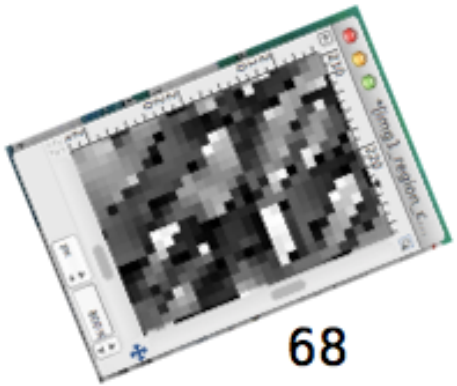
68

203

315

[3]

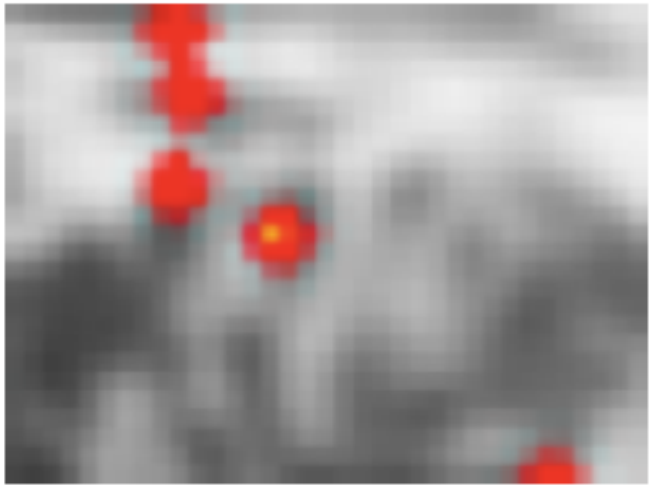
good for normalization test or preference for gradient theta (220, 220)(9, 194)



68

315

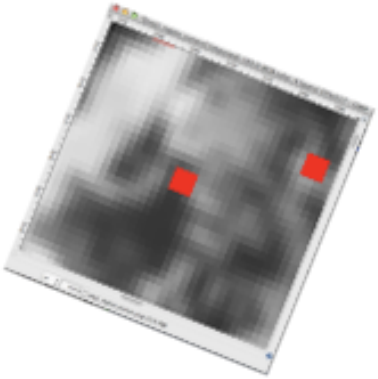
gradient theta rotated to "dominant orientation" with "dominant orientation" subtracted from theta (ref for method?)

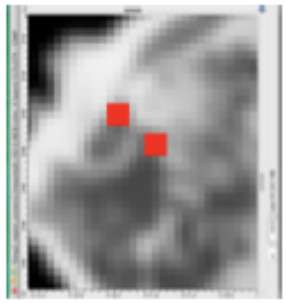


left and right clips from the Brown & Lowe showing projection and illumination differences.

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.

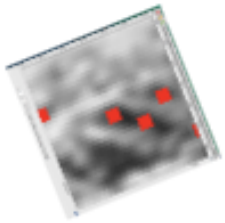
Note that the rectangular radius around the central pixel is somewhat large, 12 pixels (note too that the snapshot above is larger than a descriptor's region).

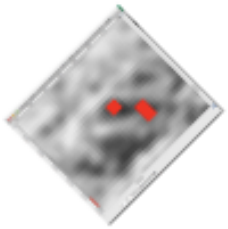


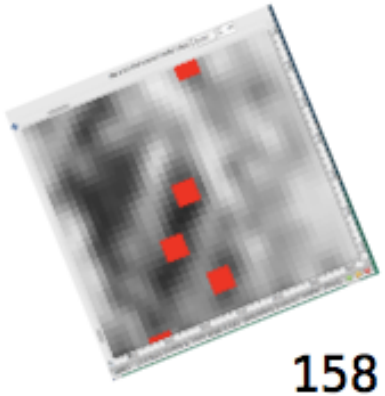


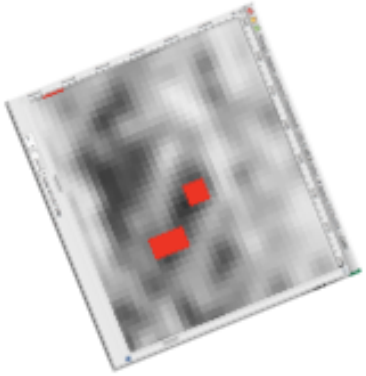
[4]

0 or 22.5 270



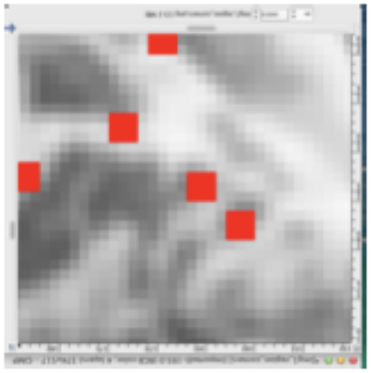


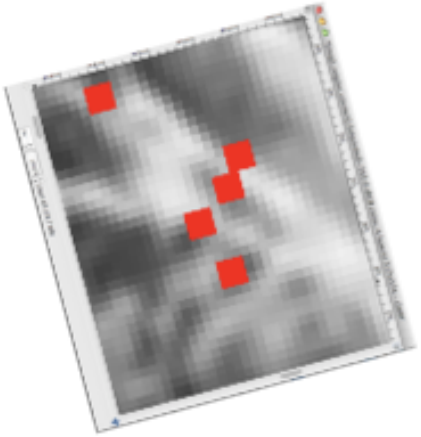




[5]

68 315 158 68





[6]

180 75?

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)
```

202.5

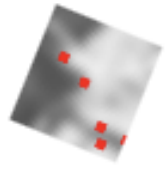
```
k[0]=0.13  x,y=(157, 466)
k[1]=0.19  x,y=(156, 466)
k[2]=0.22  x,y=(155, 466)
k[3]=0.20  x,y=(154, 466)
k[4]=0.16  x,y=(153, 467)
```

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)
```

202.5

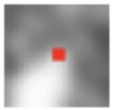


```
k[0]=0.18  x,y=(105, 298)
k[1]=0.33  x,y=(105, 297)
k[2]=0.36  x,y=(106, 296)
k[3]=0.26  x,y=(107, 296)
k[4]=0.14  x,y=(108, 296)
```

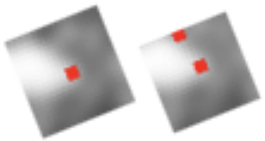
112.5

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)
```



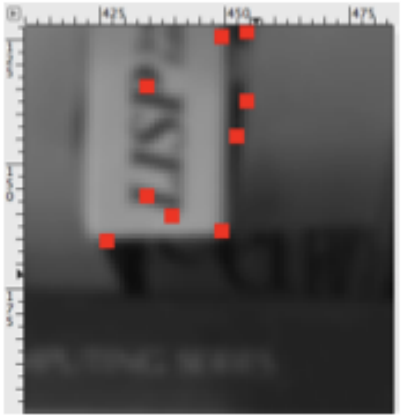
67.5

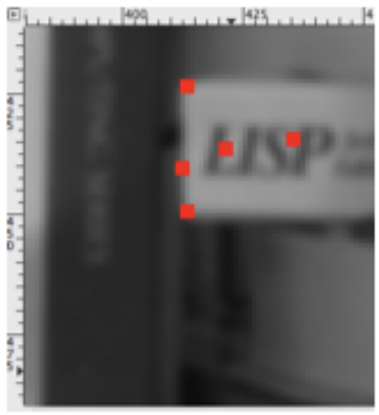


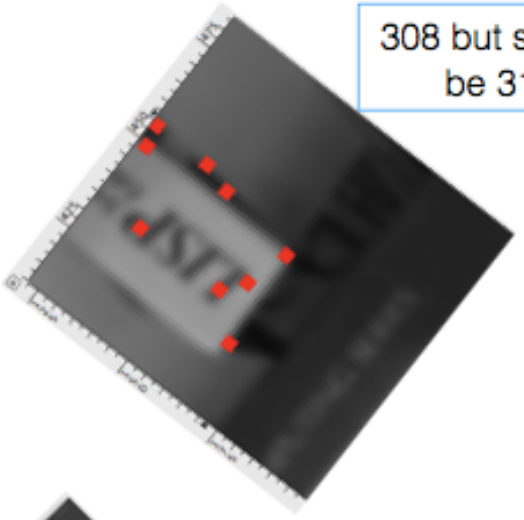
```
k[0]=0.26  x,y=(382, 448)
k[1]=0.29  x,y=(383, 447)
k[2]=0.31  x,y=(384, 446)
k[3]=0.29  x,y=(384, 445)
k[4]=0.23  x,y=(384, 444)
```

337.5

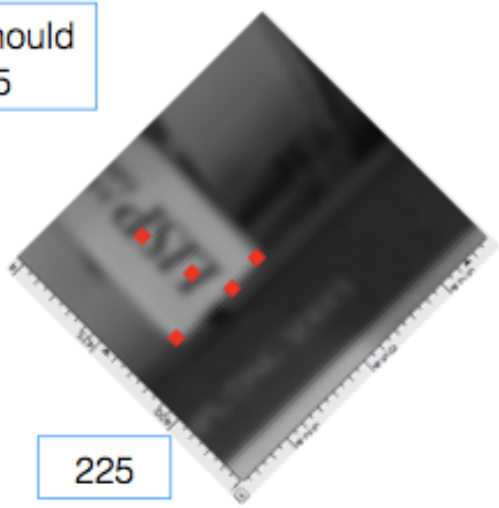
rotated



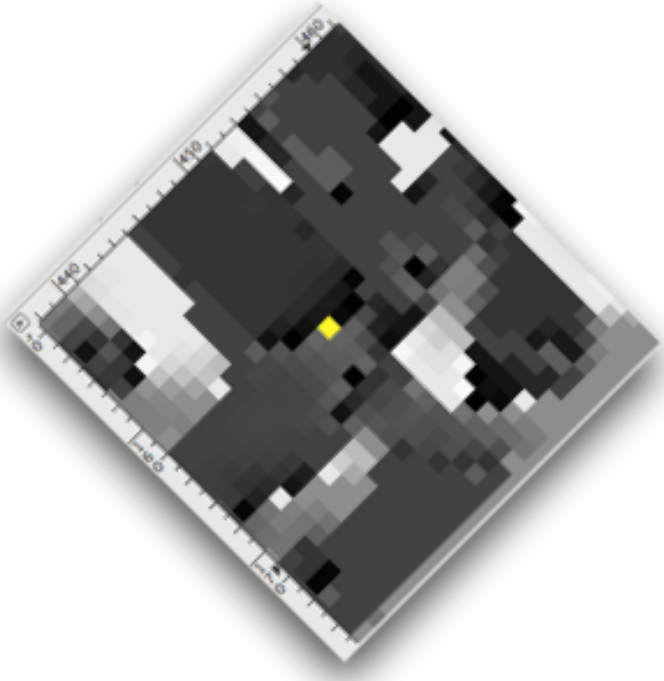


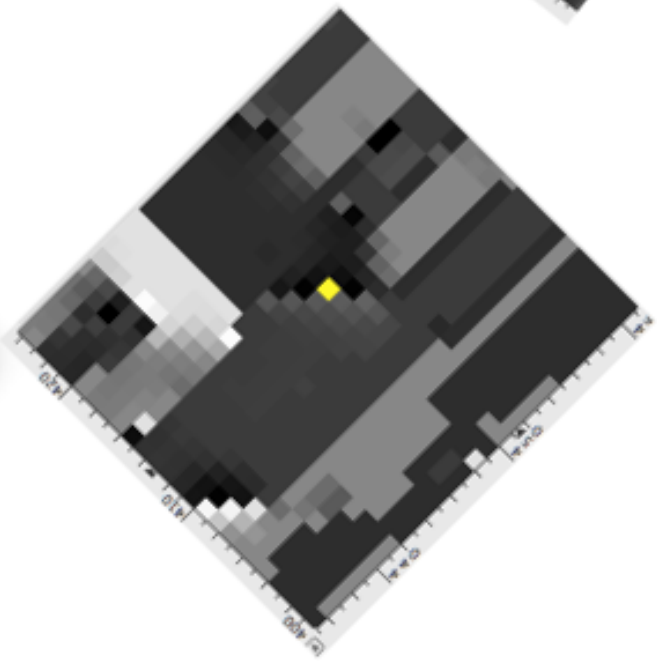


308 but should be 315



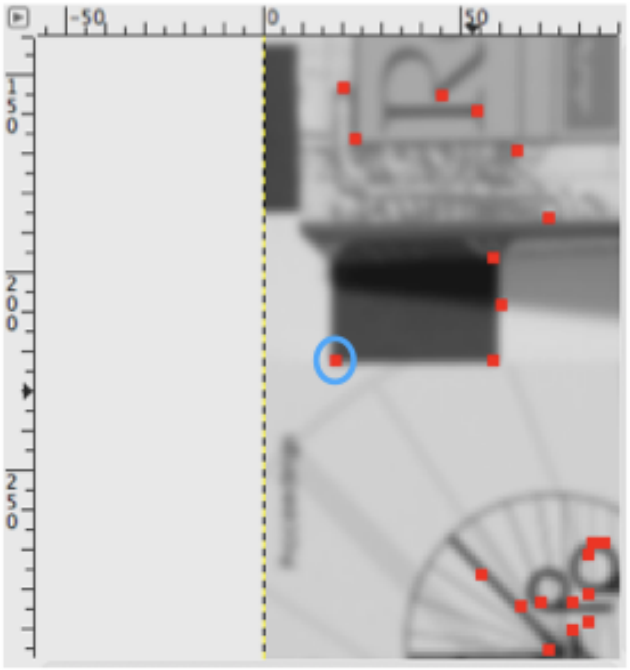
225

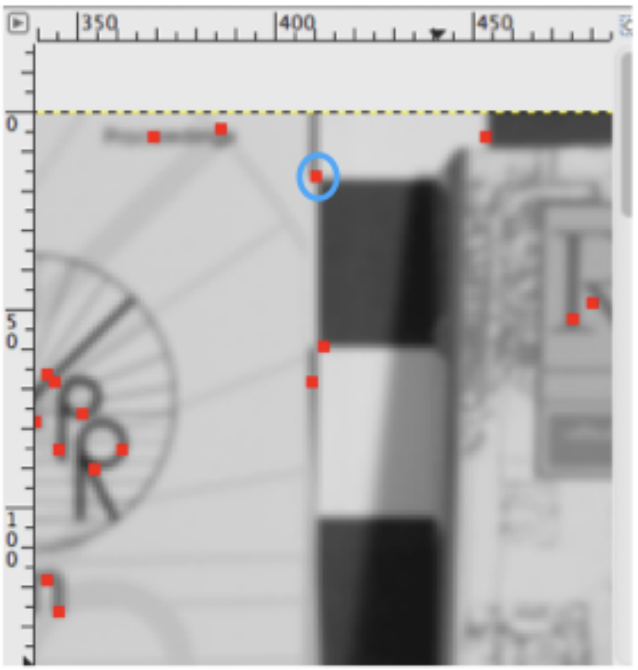




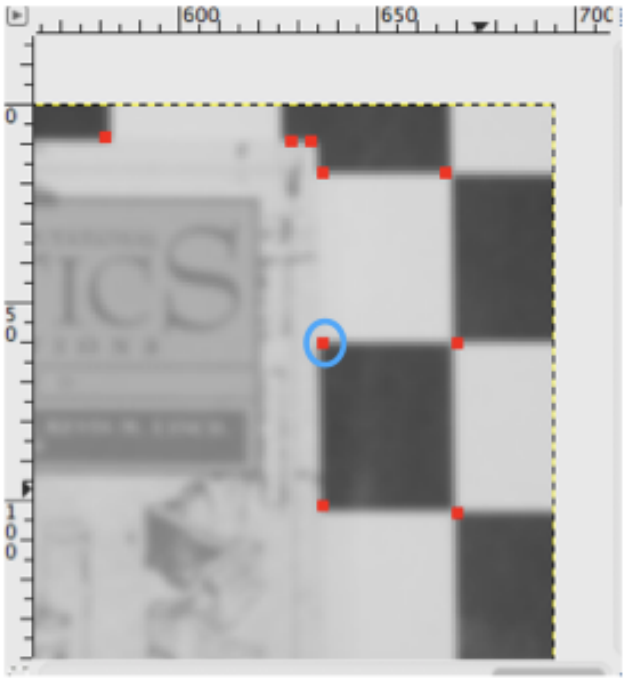
the theta descriptor SSD is larger than the error from auto-correlation. discard in favor of clearer matches and note for a future algorithm which recognizes contours and matches within…

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.





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.

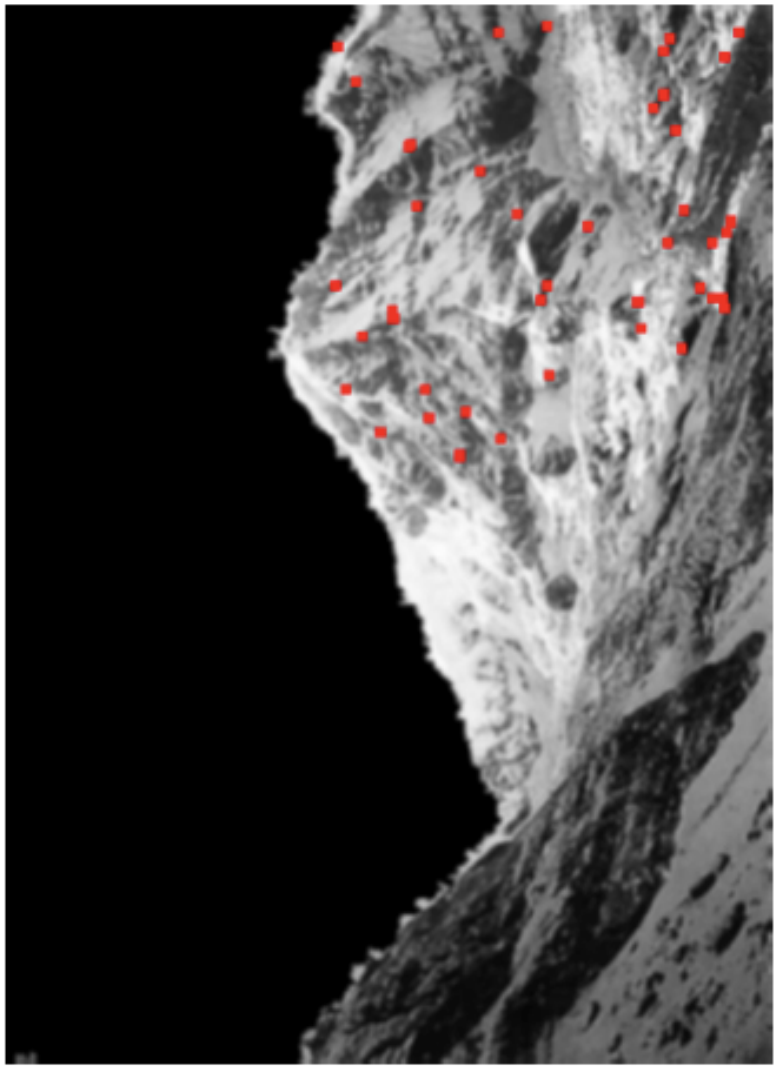
where "found" is referring to the matching algorithm to make rough correspondence lists before use of the epipolar projection solver.

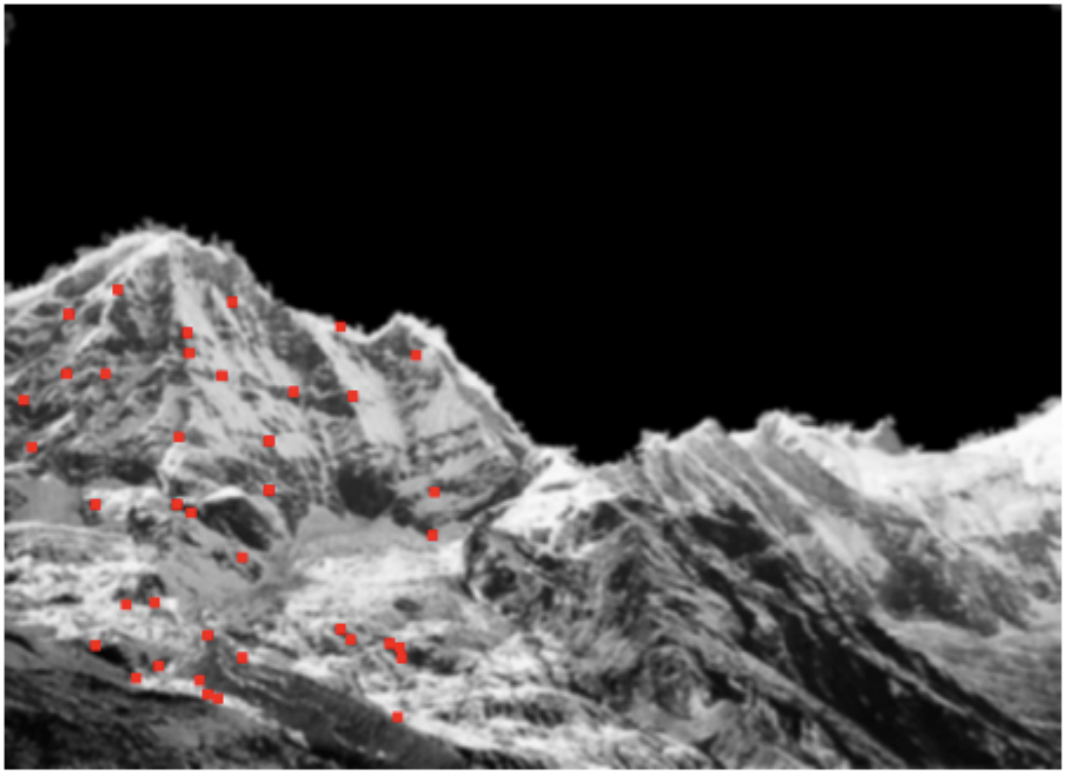
might need to consider the top k matches, especially if the image is known to contain repeated patterns/textures.

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+-55.847068786621094
translationY=-25.0+=29.851972579956055



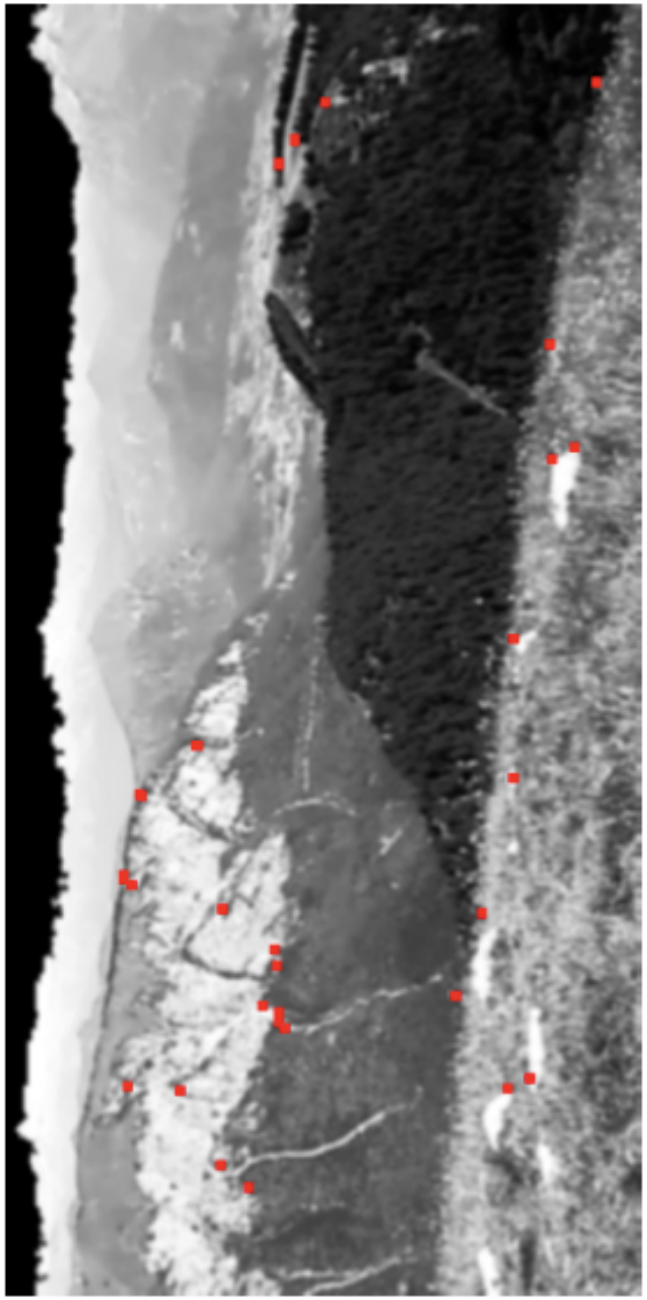


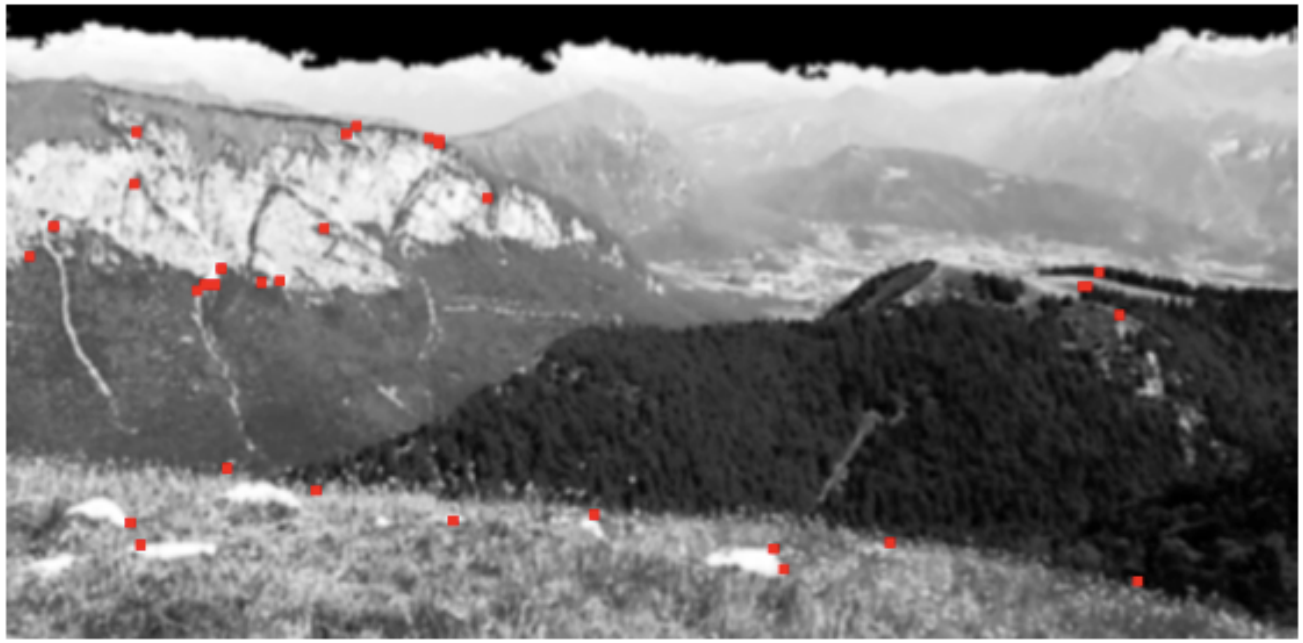
result is a list of correspondence, usable as input to the epipolar solver which uses RANSAC to discard outliers while solving the fundamental matrix.

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’, and in those cases, the correspondence list can be very fast to make by evaluating translations combined w/ features, but for images of different cameras, etc, the scale isn’t known much less the general euclidean transformation.

A scale between both images can be found if one can identify common objects present in both. That requires segmentation tailored for unique patterns in the image.

Several segmentation algorithms were tried and the best for producing several matchable blobs tended to be KMPP on greyscale intensities with k=2, followed by a DFS traversal for contiguous pixels of set size within a range.

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.

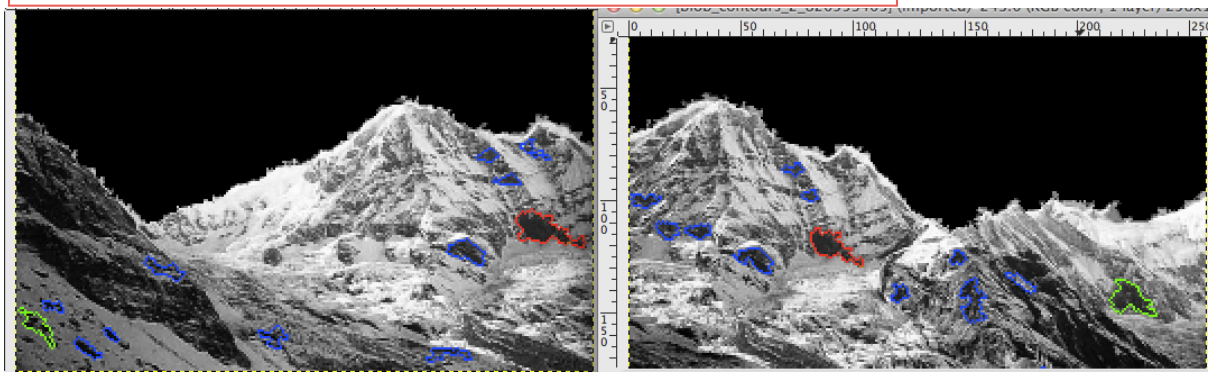
The blob finding performed on images binned to < 300 x 300 and on the full images seems to produce feasible matching features even though there are few.

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

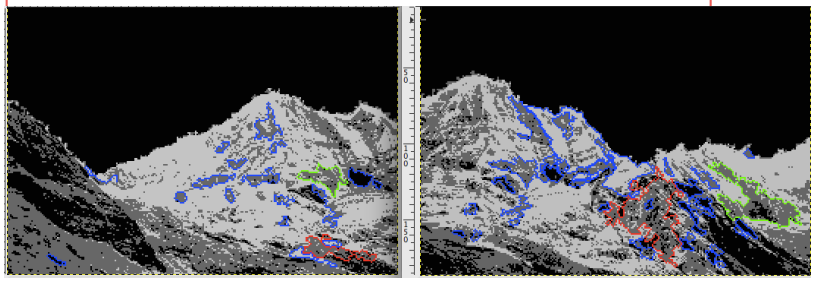
Once scale is calculated from blobs, ***features*** are used to create correspondence lists for input to epipolar projection solver (see previous pages).

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

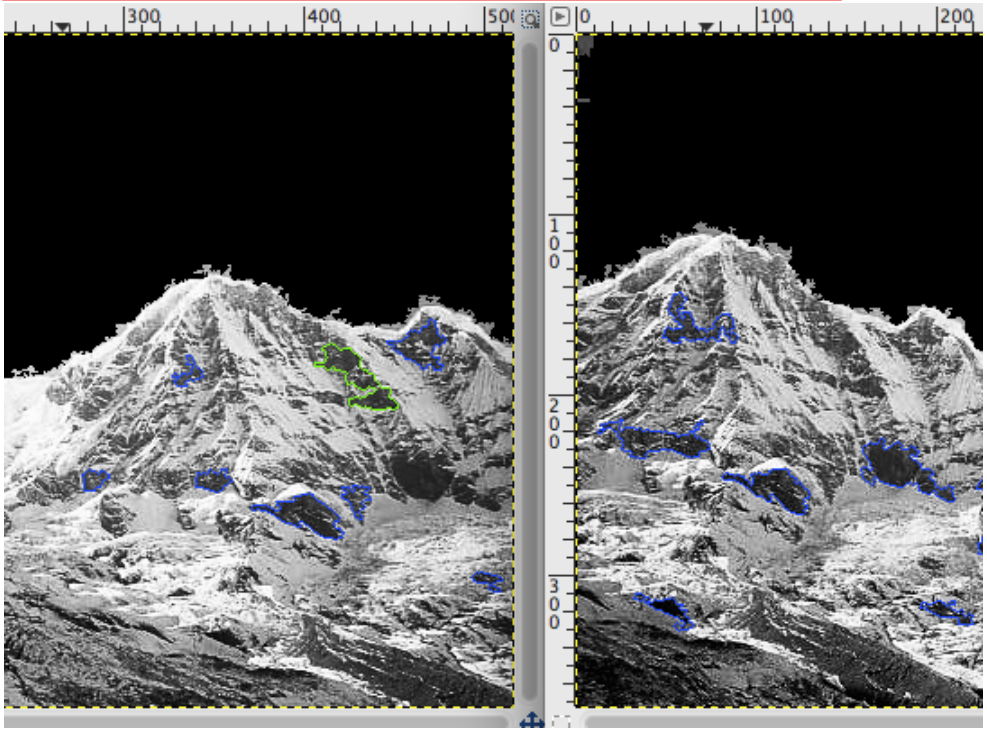
k=2 segmentation blobs, binned to < 300x300



k=3 segmentation blobs, < 300x300



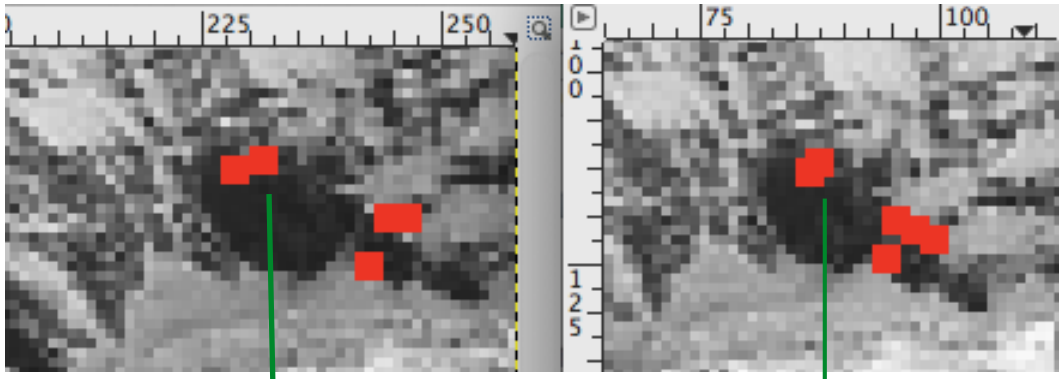
k=2 segmentation blobs, image not binned



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:



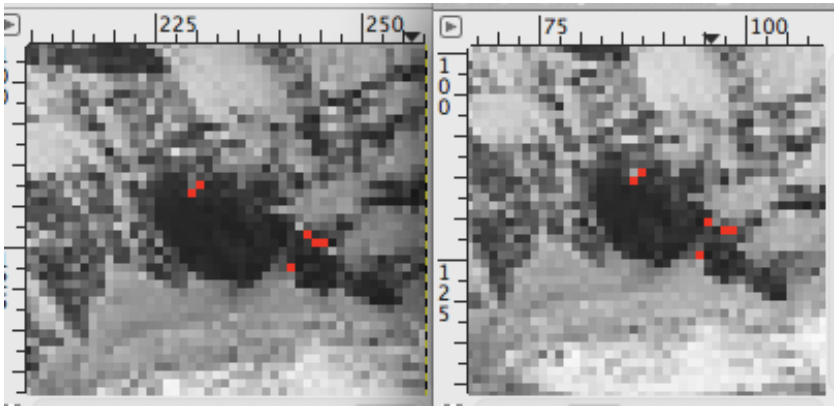

```
[0](236,123) [0](90,120) cost=1.0 scale=0.86  nMatched=4
(245,123) theta1=90    (95,120) theta2=203  intensity=4618.6(4808.2),
(246,123) theta1=45    (97,121) theta2=90
(242,128) theta1=23    (94,124) theta2=225
(228,118) theta1=203   (86,115) theta2=270  intensity=2531.4(2671.8),
(231,117) theta1=135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



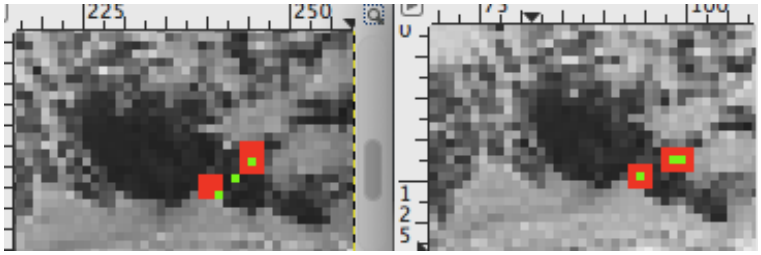

```
[0] [0] dither=2, using previous best rotations for all peaks
(243,123) (95,120) intSqDiff=598.3(2155.0)
(244,124) (97,121) intSqDiff=452.4(2820.2)
(241,127) (94,124) intSqDiff=273.9(4022.0)
(229,118) (86,115) intSqDiff=179.3(1226.4)
(230,117) (87,114) intSqDiff=175.5(1835.1)
(245,124) (98,121) intSqDiff=316.7(2304.6) 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
```

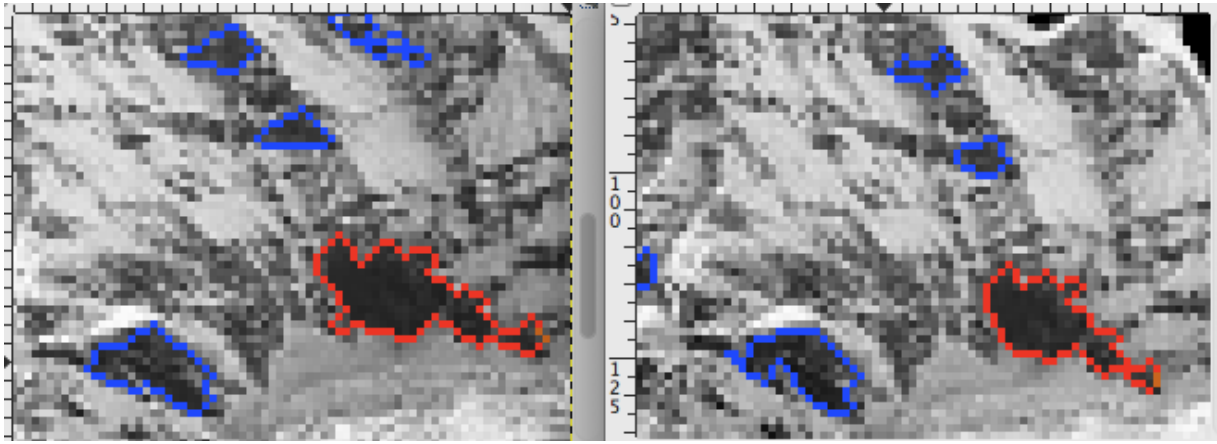
The improvement works pretty well.
The correct match has smaller sum square of differences now making the true match of [0][0] easier to distinguish from the false match of [0][1].

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).

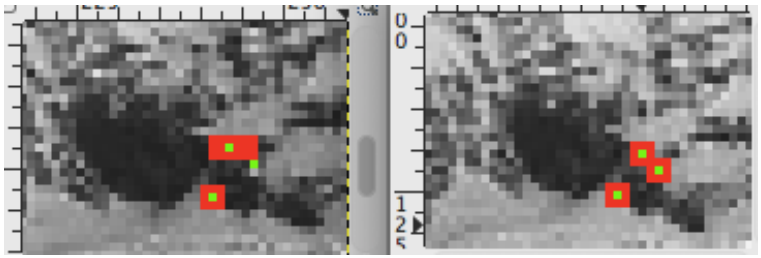
Red are the contour matched points. **Light green** are the feature improved locations.

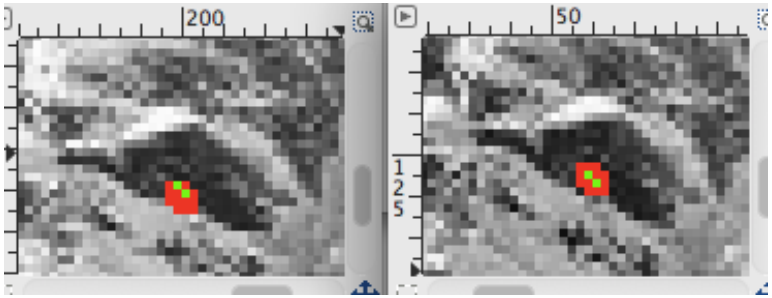
before image guided corrections:

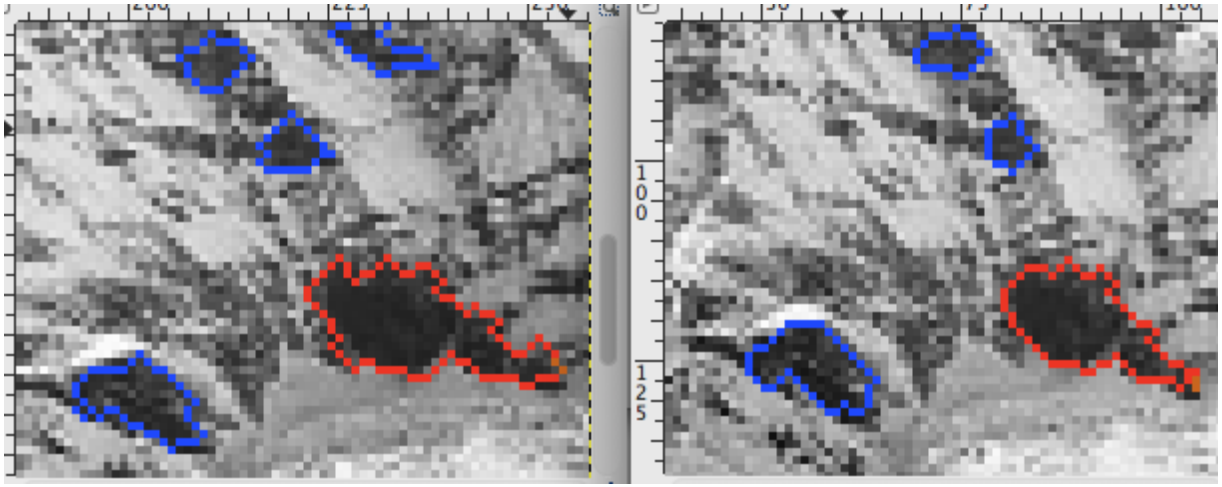




after image guided corrections:

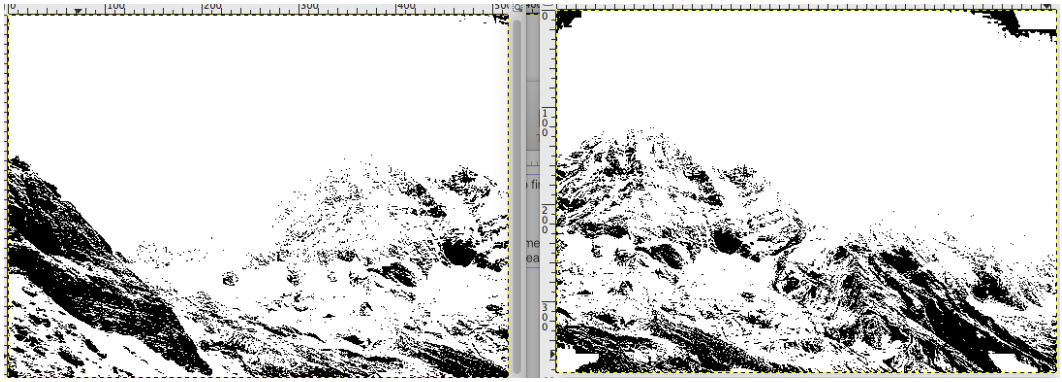




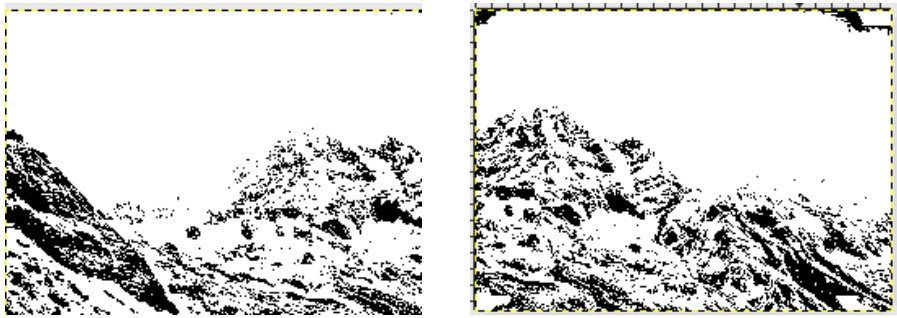


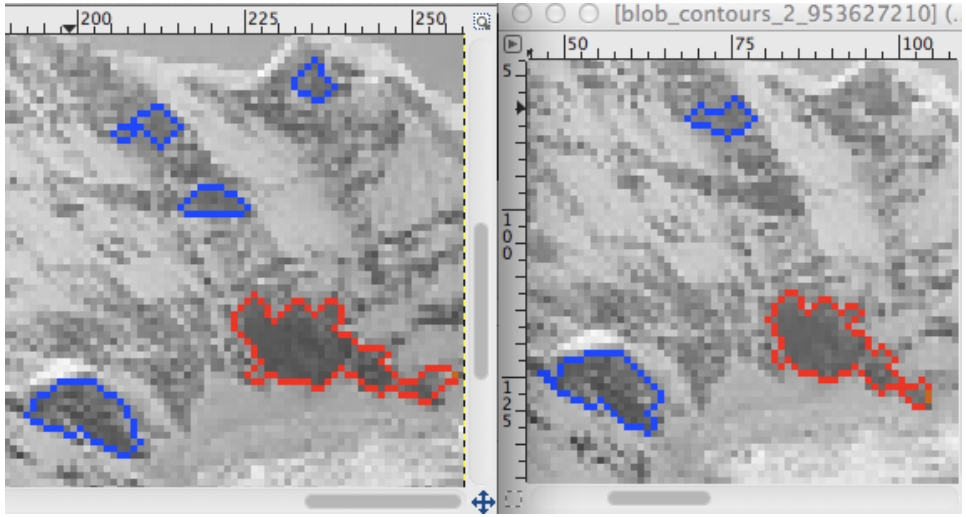
alternate (faster) way to process image to find same blobs.
no sky subtraction needed for pre-processing.

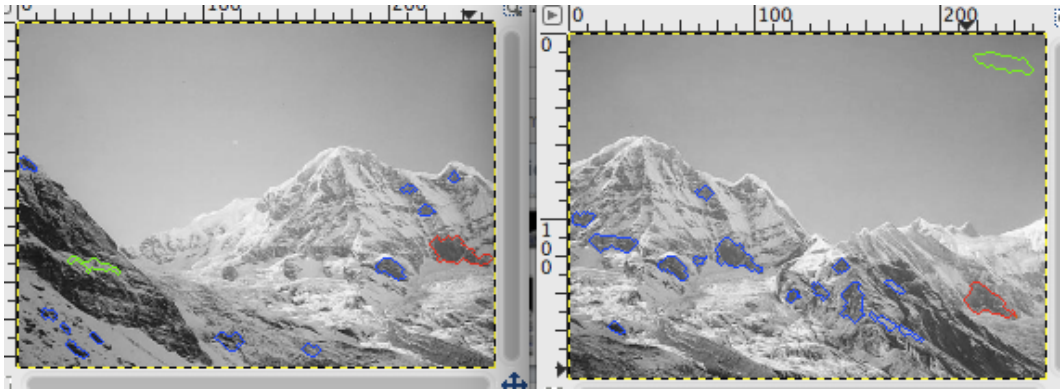
```
img0 = img.copyToGreyscale();
imageProcessor.applyImageSegmentation(img0, 2);
imageProcessor.applyAdaptiveMeanThresholding(img0, 20);
```



same, but binned, and
adapMeanThr=20/binFactor=20/3

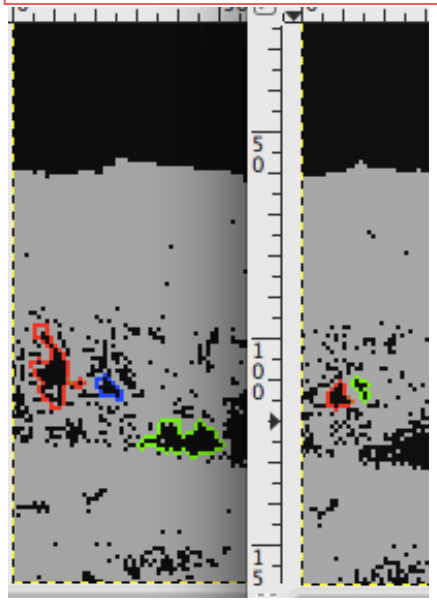




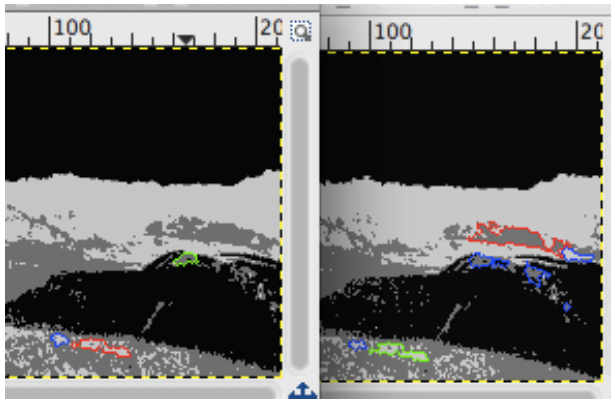


A look at blob contours (to use for scale solutions)

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



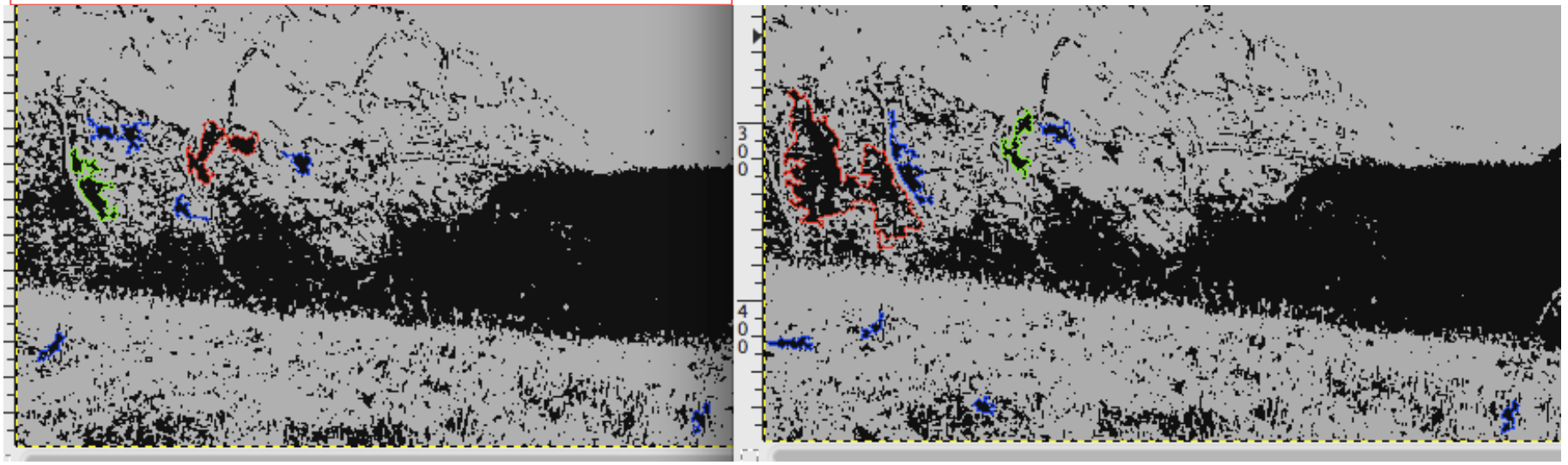
k=3 segmentation blobs, < 300x300, no histeq

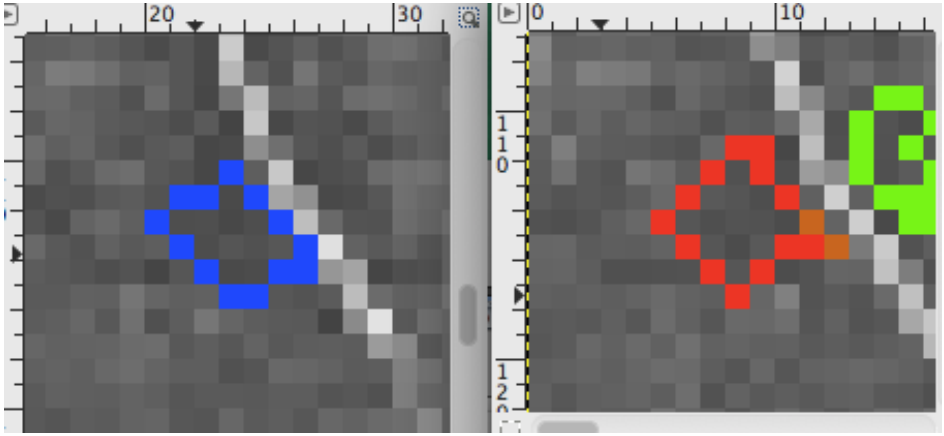


binary segmentation on binned images and images not binned is not the best solution for these images. still exploring a better means.

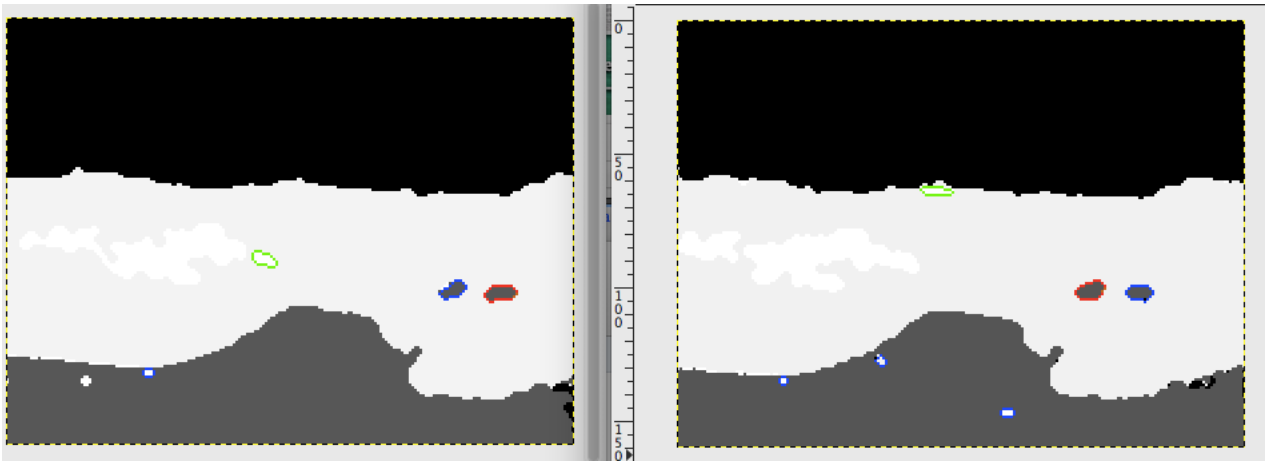
k=3 segmentation blobs, image not binned
has about 4 strong blobs in common

k=2 segmentation blobs, image not binned





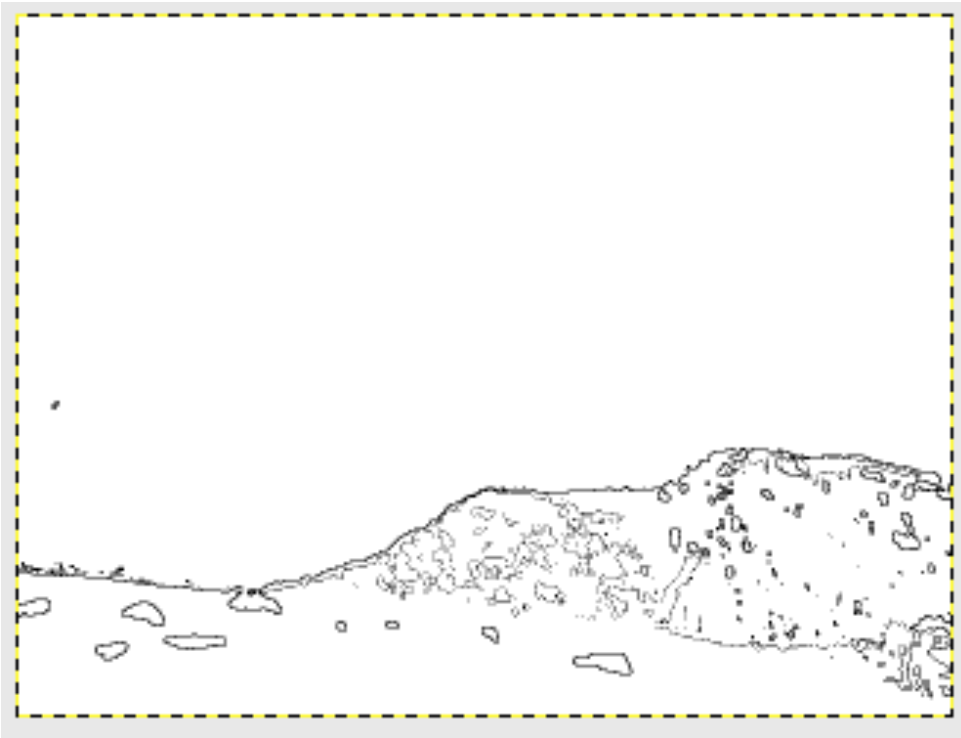
inflection points not always best choice for points of interest. For example, these small rectangles have small scale space contours that are revisited and included if larger contours (from closed curves) are not found.

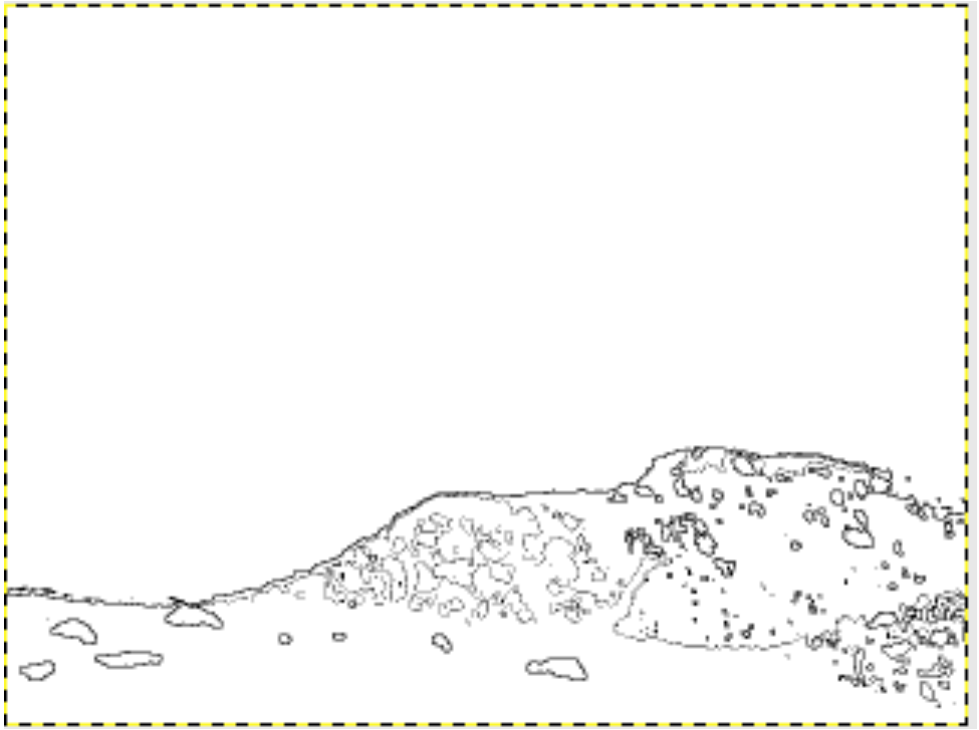


There are also other characteristics that can be seen with segmentation that would make good matching features. This would make a good image to make corners from. The unsegmented unbinned image is a very busy high resolution image with details such as grass texture that make many edges and corners.

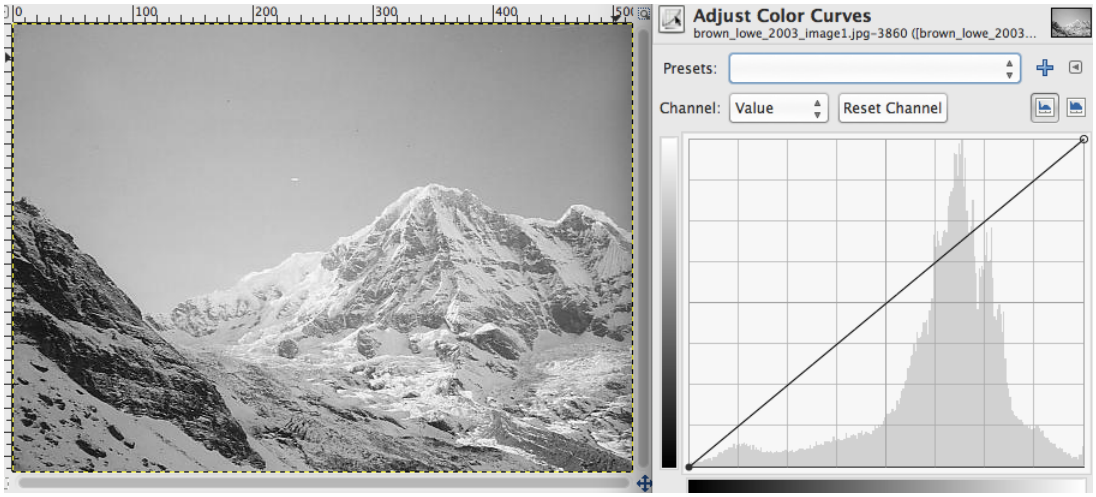
The scale solution still needs closed contours though so need to consider the small circles that have small scale space contours if larger contours (from closed curves) are not found.

Another approach: Color segmentation followed by adaptive mean thresholding removes the grass as noise and sky variation. The foreground rocks are matchable, but there are many small blobs too. Those will not pass the number of points low filter.

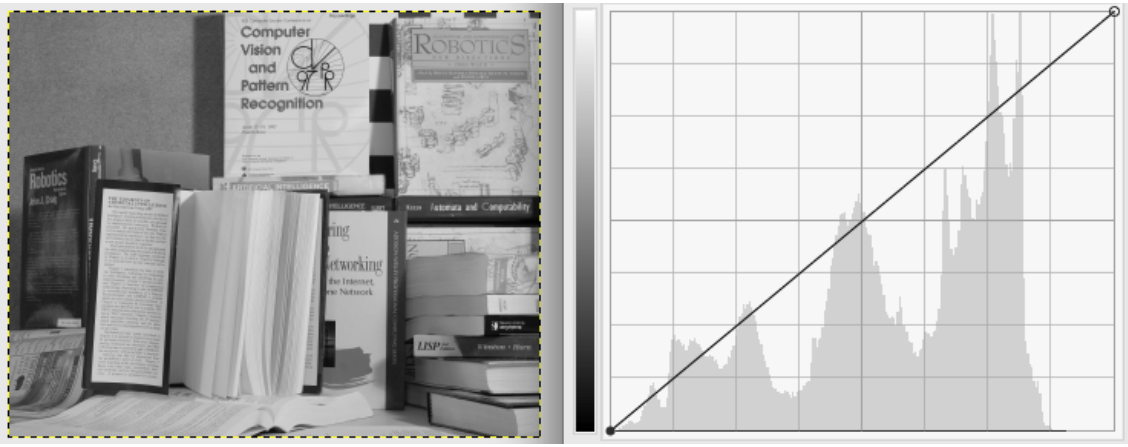




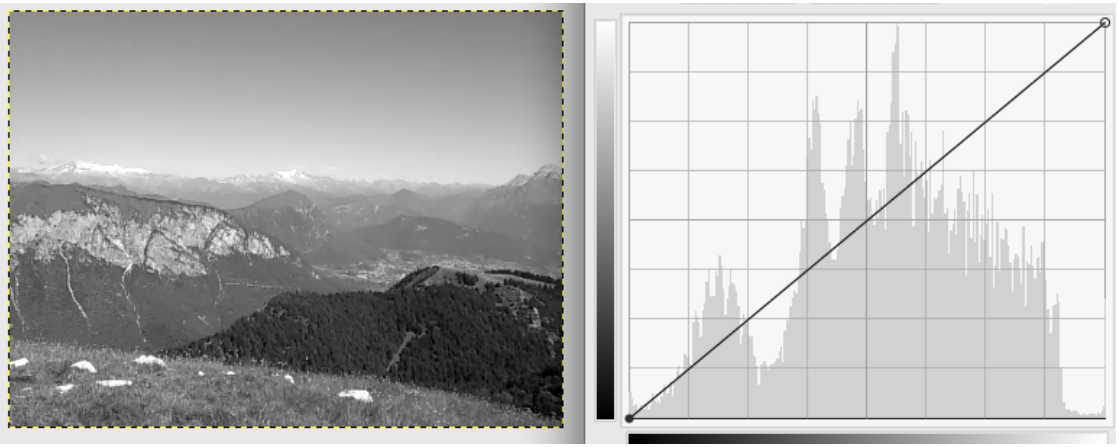
When most of pixels are bright, prefer to use binary segmentation and binning to find dark blobs.



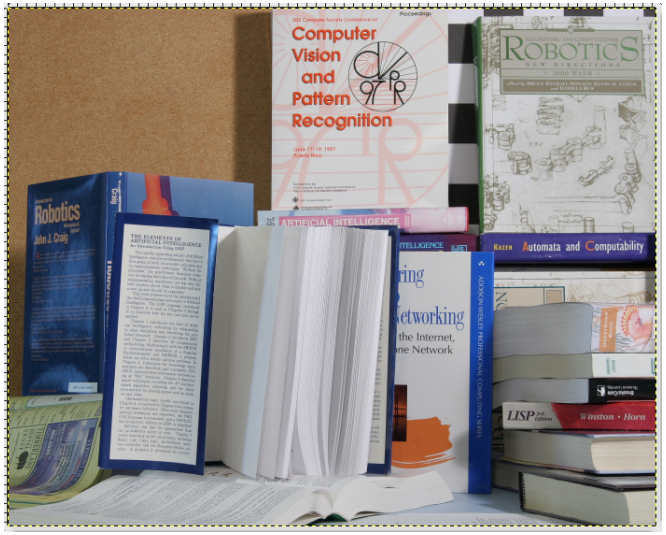

```
stats=min=0.0
    [junit] max=255.0
    [junit] mean=159.15577697753906
    [junit] median=163.0
    [junit] mode=165.19515991210938
    [junit] quartiles=[145.0, 163.0, 180.0, 255.0]
    [junit] histogram=histogram=[(9.717362, 116) (29.152088, 780)
(48.58681, 2144) (68.02154, 5951) (87.45626, 6139) (106.89098, 8183)
(126.325714, 13823) (145.76044, 36746) (165.19516, 57768) (184.62988,
37465) (204.0646, 11988) (223.49933, 7597) (242.93407, 4009) ]
    [junit] histogram errors=[(9.717363, 0.77178496) (9.717363,
0.20207894) (9.717363, 0.15404044) (9.717363, 0.10978928) (9.717363,
0.12205033) (9.717363, 0.116789326) (9.717363, 0.099361) (9.717363,
0.06792254) (9.717363, 0.058328137) (9.717363, 0.07409418) (9.717363,
0.1321732) (9.717363, 0.17136808) (9.717363, 0.24313246) ]
histAreas=[0.23737653, 0.7626235]
```

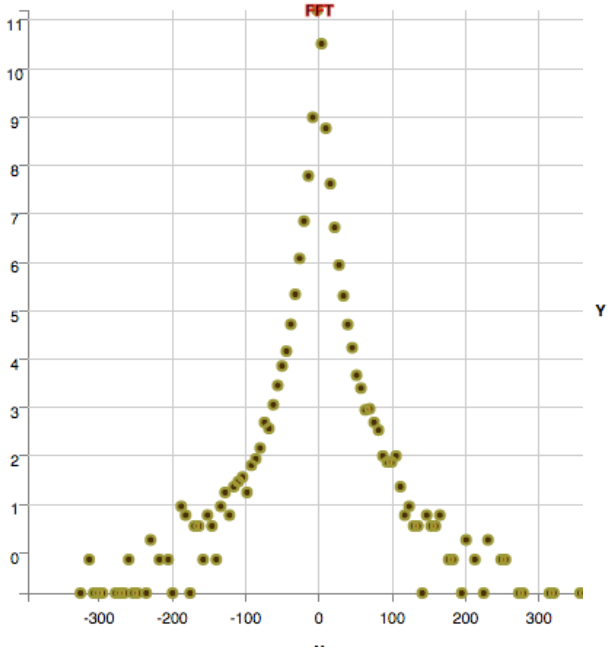


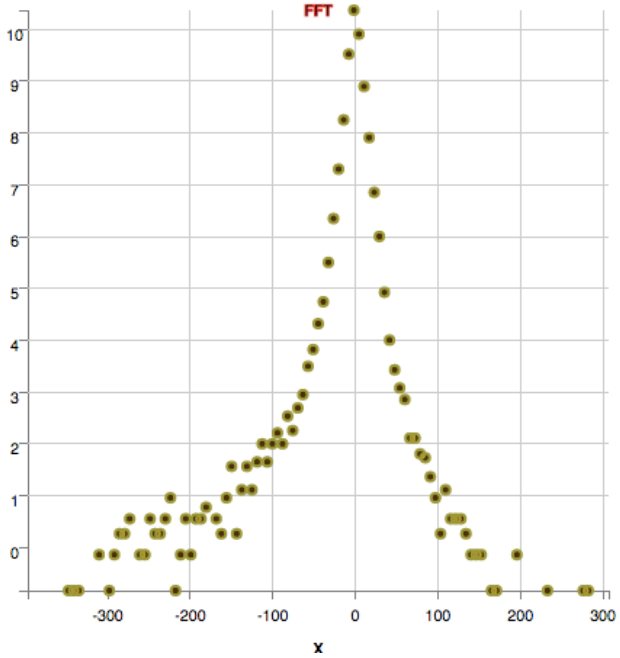

```
stats=min=0.0
    [junit] max=230.0
    [junit] mean=142.4168701171875
    [junit] median=150.0
    [junit] mode=193.89910888671875
    [junit] quartiles=[113.0, 150.0, 190.0, 230.0]
    [junit] histogram=histogram=[(9.233291, 1934) (27.699871,
11960) (46.166454, 21237) (64.63303, 25864) (83.09962, 16128)
(101.5662, 15522) (120.032776, 58025) (138.49936, 38071)
(156.96594, 25586) (175.43253, 55022) (193.89911, 82024)
(212.36569, 34202) (230.83226, 150) ]
    [junit] histogram errors=[(9.23329, 0.129593) (9.23329,
0.053420994) (9.23329, 0.050305214) (9.23329, 0.054534614)
(9.23329, 0.07460825) (9.23329, 0.08468471) (9.23329,
0.049914666) (9.23329, 0.063751675) (9.23329, 0.08130212)
(9.23329, 0.06033505) (9.23329, 0.052829705) (9.23329,
0.08098979) (9.23329, 1.1787614) ]
histAreas=[0.31374013, 0.68625987]
```

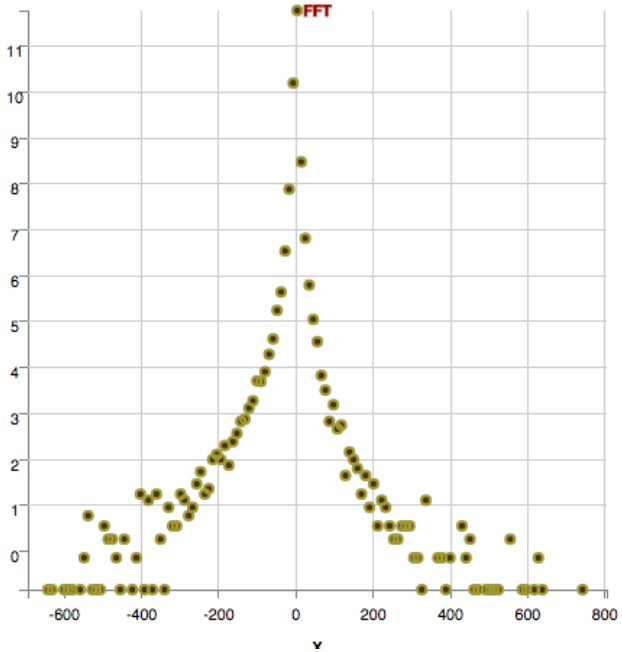


```
stats=min=0.0
    [junit] max=255.0
    [junit] mean=130.99832153320312
    [junit] median=134.0
    [junit] mode=140.81146240234375
    [junit] quartiles=[92.0, 134.0, 173.0, 255.0]
    [junit] histogram=histogram=[(9.38743, 2005) (28.16229, 10477)
(46.93715, 25751) (65.71201, 10222) (84.48687, 33586) (103.26173,
32719) (122.03659, 32886) (140.81146, 42862) (159.58632, 33073)
(178.36118, 31383) (197.13603, 25580) (215.9109, 21222)
(234.68576, 3989) ]
    [junit] histogram errors=[(9.38743, 0.1964549) (9.38743,
0.054399375) (9.38743, 0.04402962) (9.38743, 0.08140163) (9.38743,
0.052351285) (9.38743, 0.057749763) (9.38743, 0.061792303)
(9.38743, 0.05860566) (9.38743, 0.07022972) (9.38743, 0.07583307)
(9.38743, 0.08779022) (9.38743, 0.10032542) (9.38743, 0.2337324) ]
histAreas=[0.5551447, 0.44485524]
```





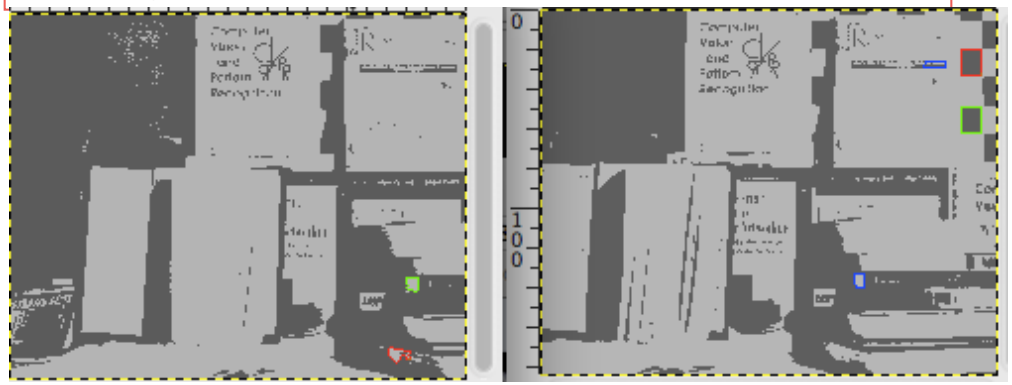




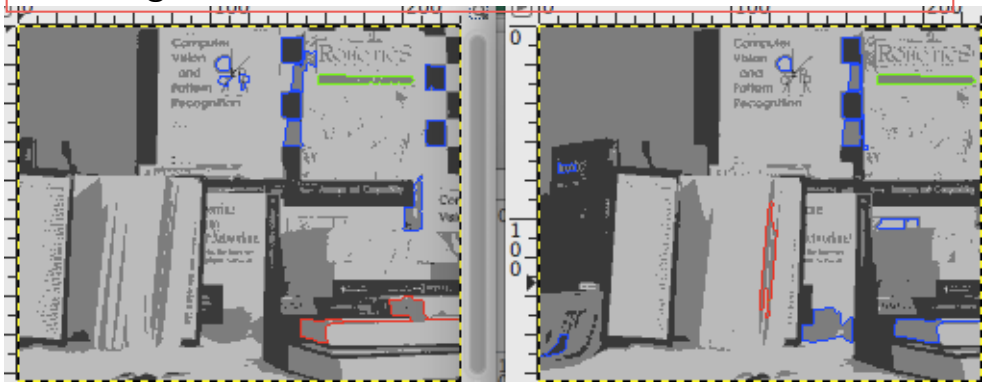
FFT of brightness, logarithm of histogram to show spatial frequency (lots of added components…)

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

k=2 segmentation blobs, binned to < 300x300

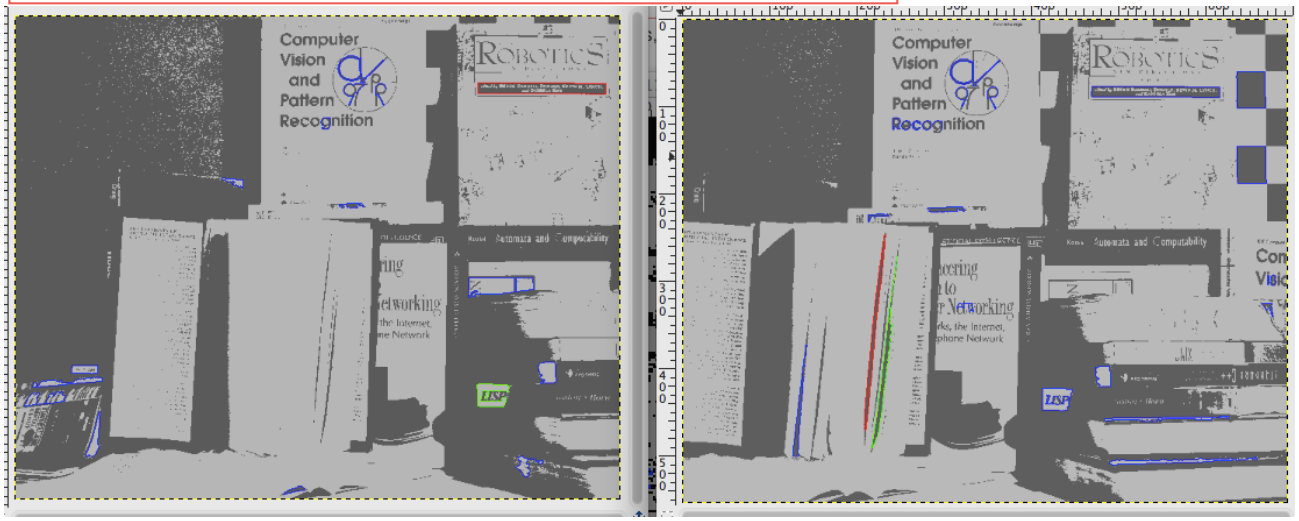


k=3 segmentation blobs, binned to < 300x300



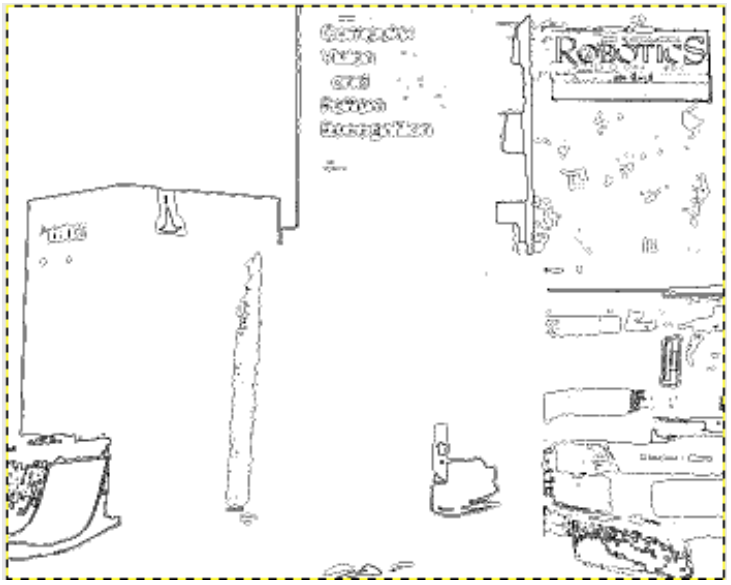
In summary with other images, a feasible approach to using blob contours to solve for *scale* may be to use binary segmentation (k=2) on binned images and images not binned,and combine the results w/ expectation that there may only be one or two blobs in the solutions.
In this case, the number of possible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

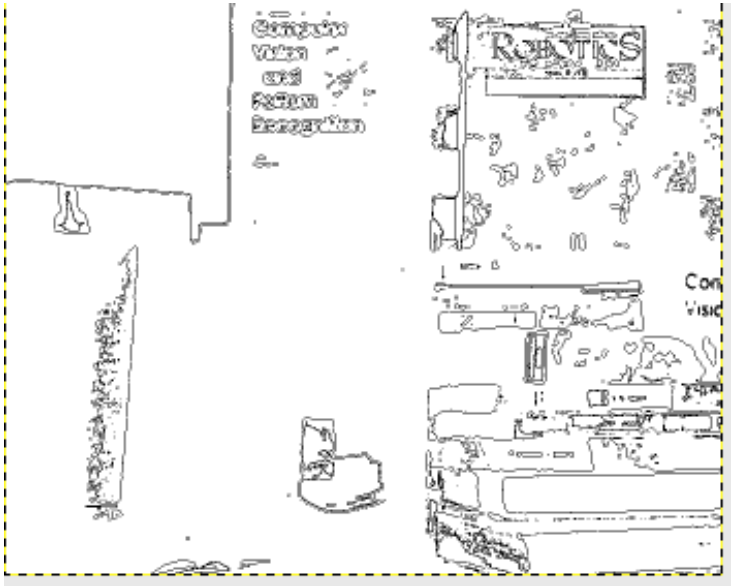
k=2 segmentation blobs, image not binned



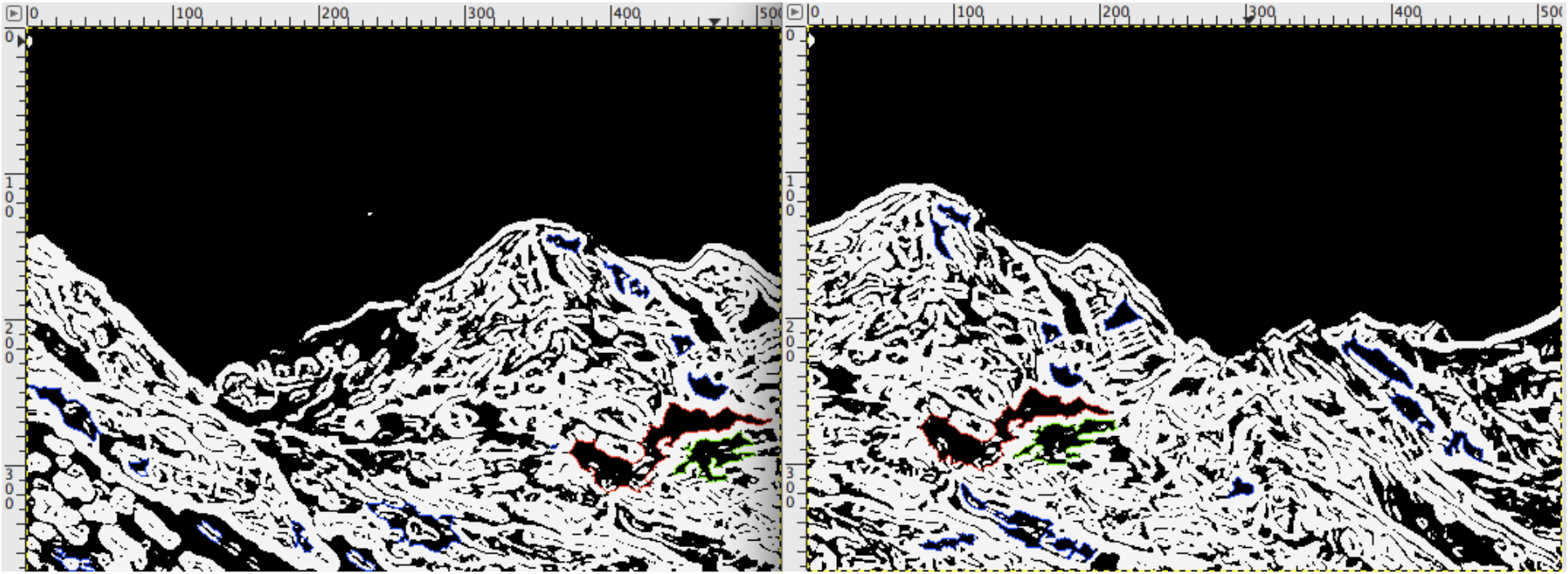
Adaptive Thresholding works well for these images with different lighting and details such as text.

If color segmentation is used first, can reduce the noise to these:

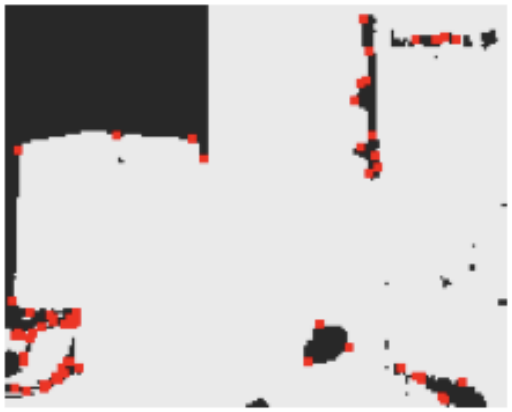


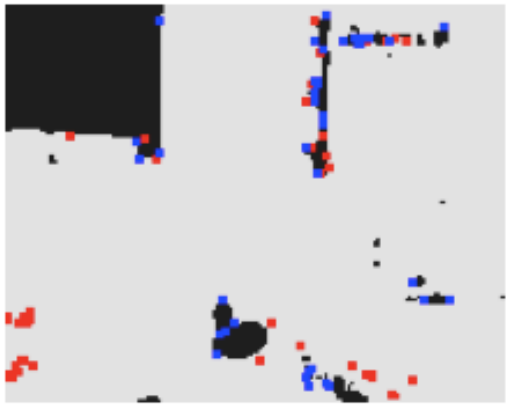


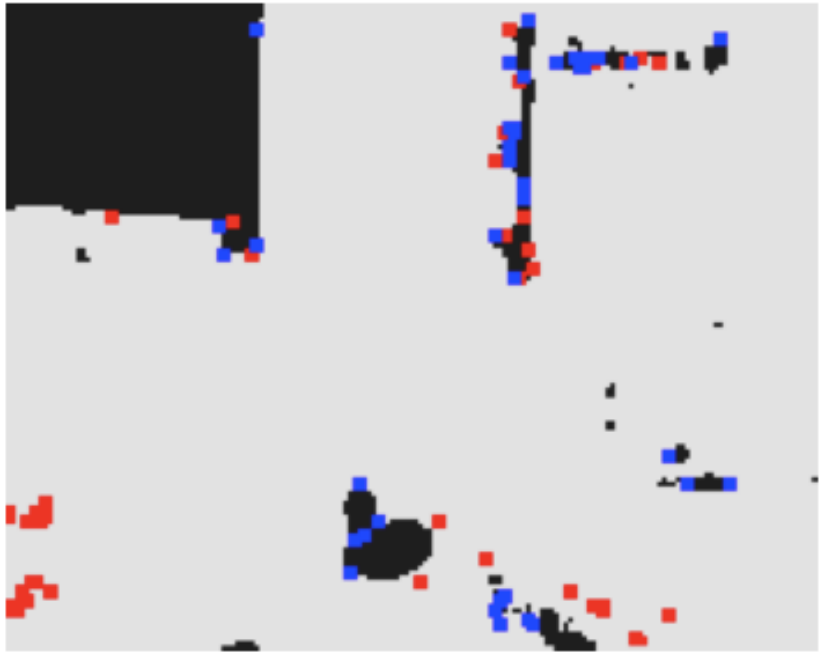
```
segImg = imgGrey.copyImage();
CannyEdgeFilter filter = new CannyEdgeFilter();
CannyEdgeFilterSettings settings = new CannyEdgeFilterSettings();
settings.setUseOutdoorMode();
filter.setSetters(settings);
filter.applyFilter(segImg);
EdgeFilterProducts edgeFilterProducts = filter.getEdgeFilterProducts();
segImg = edgeFilterProducts.getGradientXY();
imageProcessor.applyImageSegmentation(segImg, 2);
for (int i = 0; i < segImg.getWidth(); ++i) {
    for (int j = 0; j < segImg.getHeight(); ++j) {
        int v = 100*segImg.getValue(i, j);
        segImg.setValue(i, j, v);
    }
}
```



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).

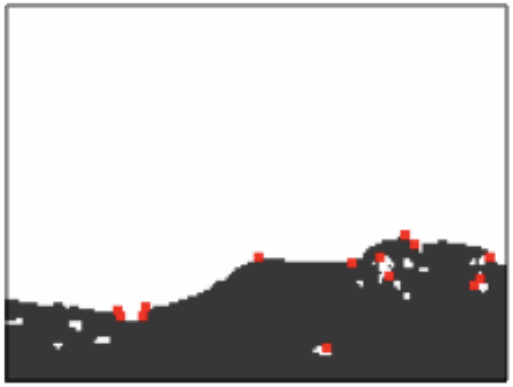


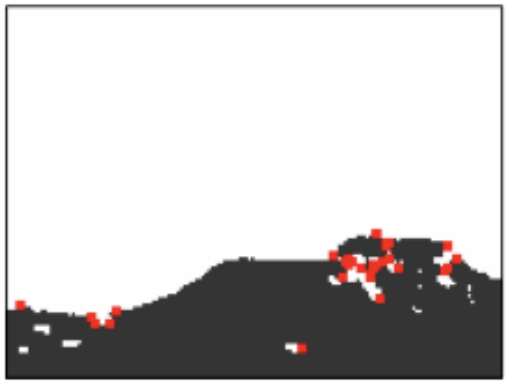


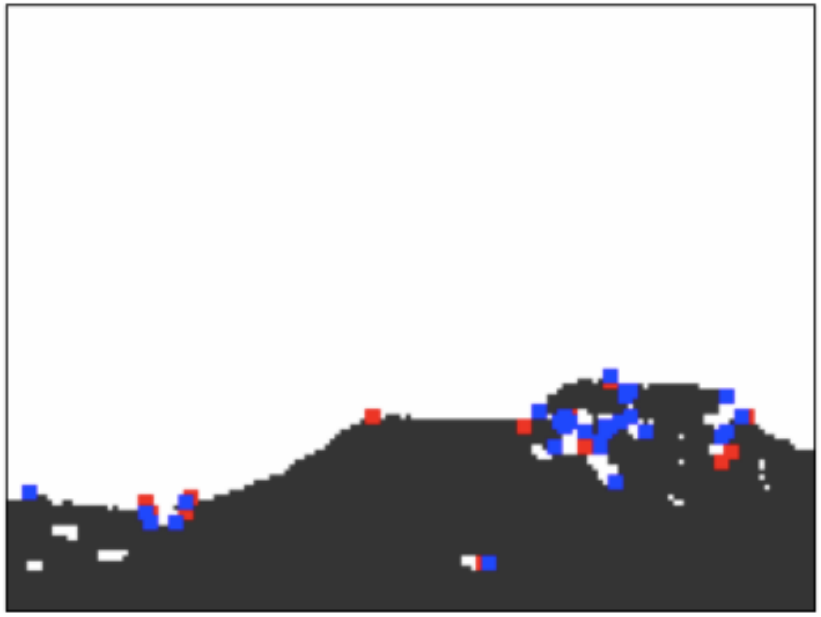


For the stereoscopic above, there's more than translation so this can only be the start of the solution

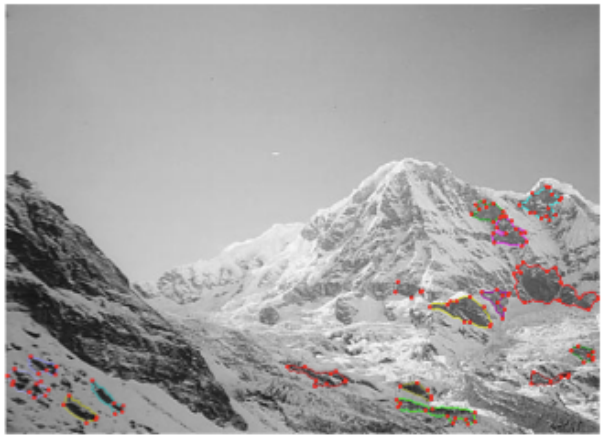
previous notes from a look at binning and segmentation

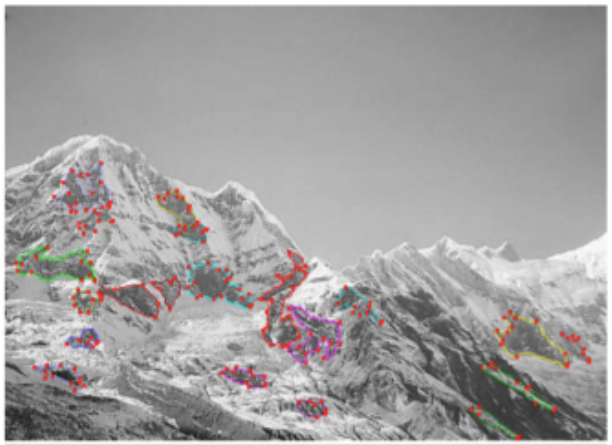






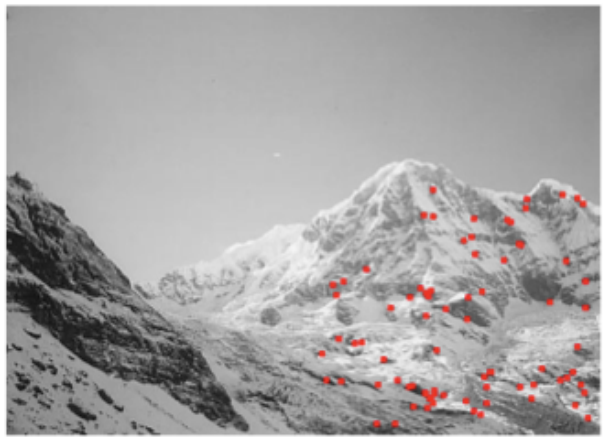
current summary of calculating euclidean transformation with blobs, then creating a correspondence list for the epipolar solver.

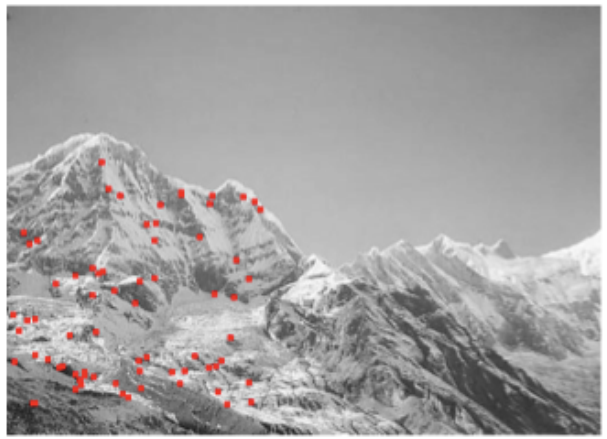




corners of blobs in kmpp segmentation.

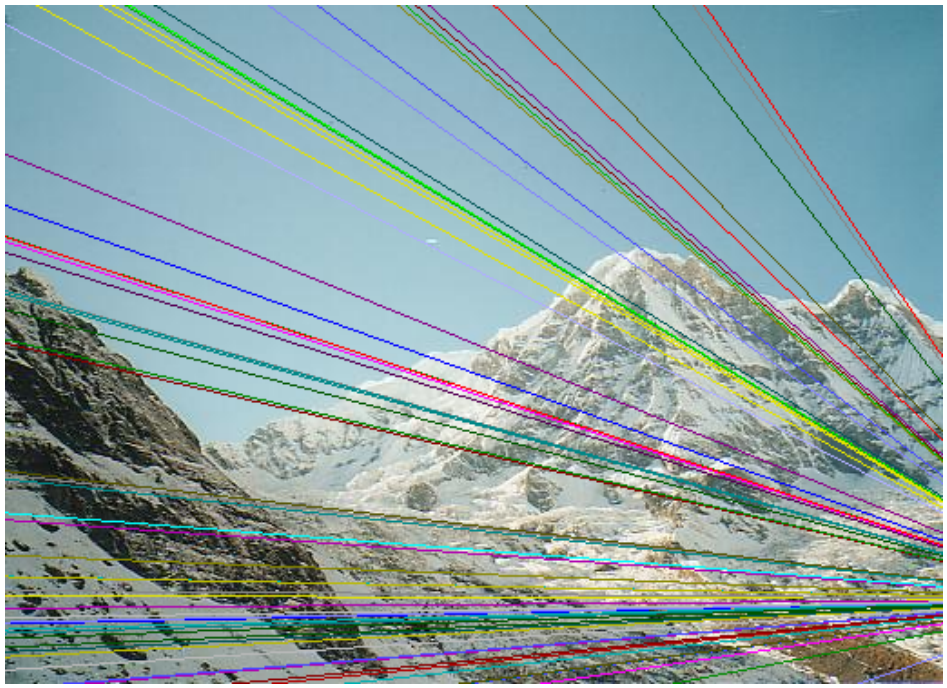
matched corners from euclidean solution

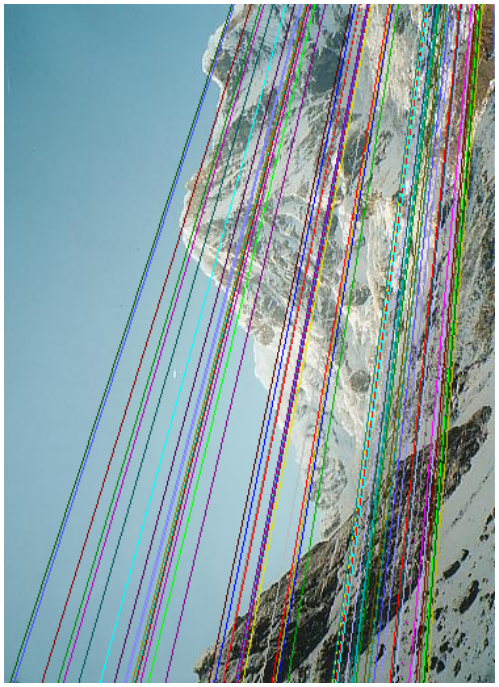


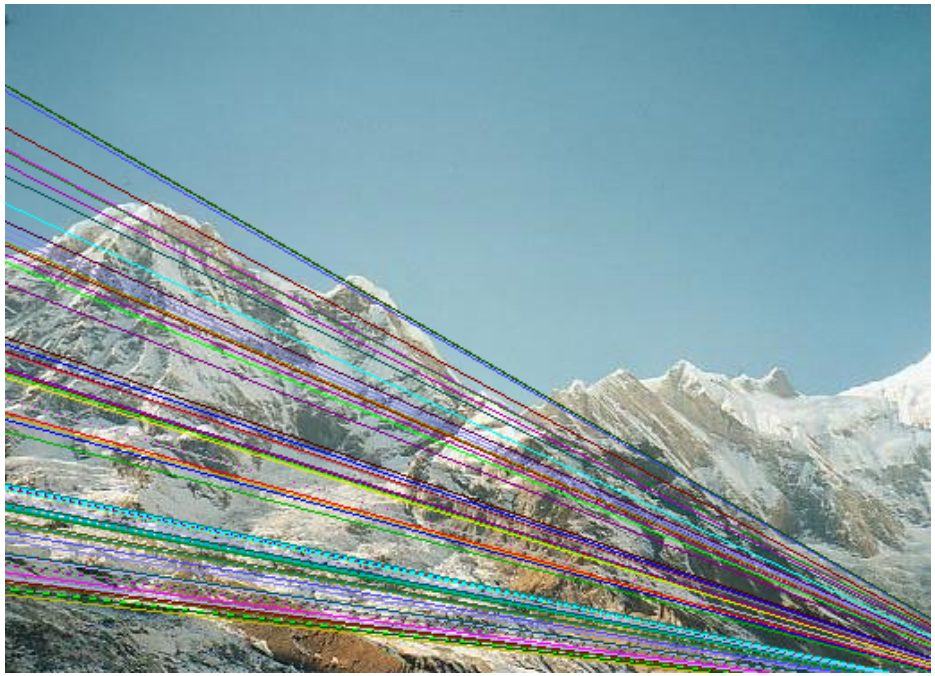


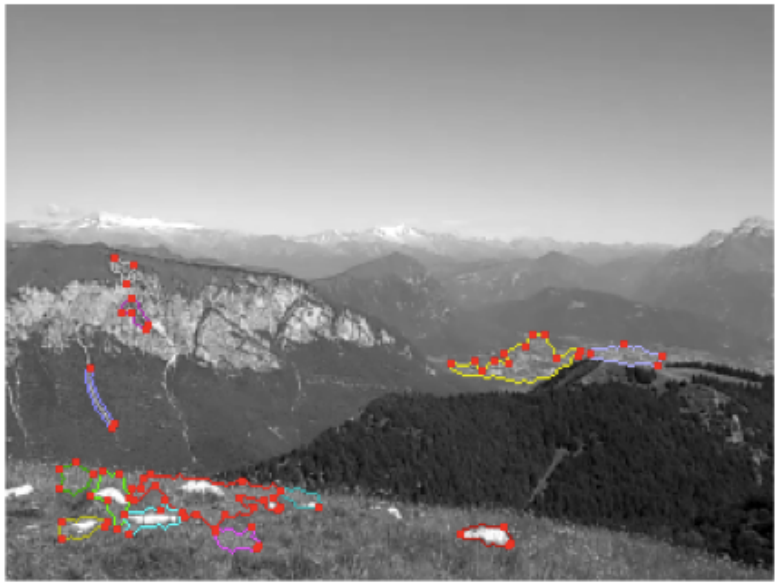
matched corners from canny edge corners found with above blob calculated euclidean solution

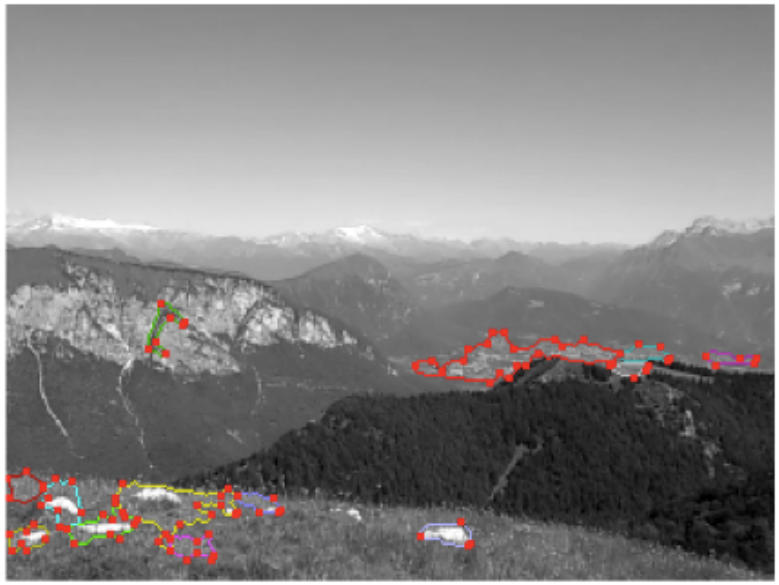
(before RANSAC and epipolar solver)



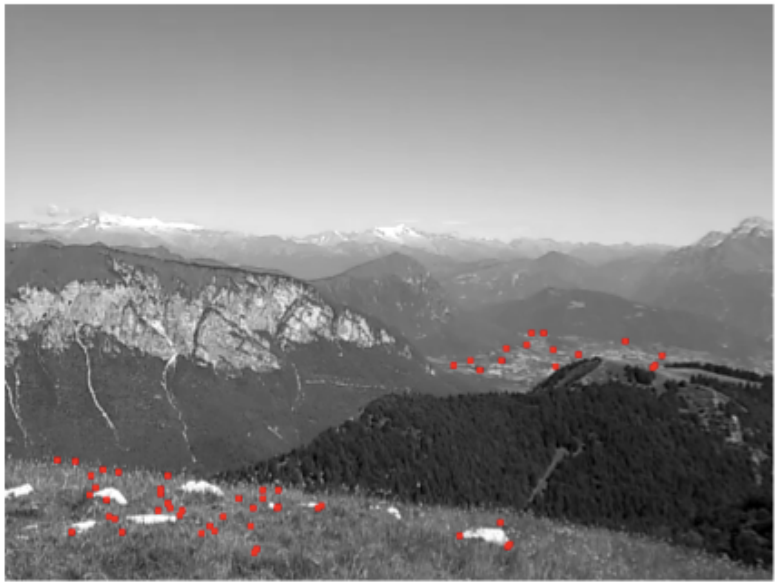


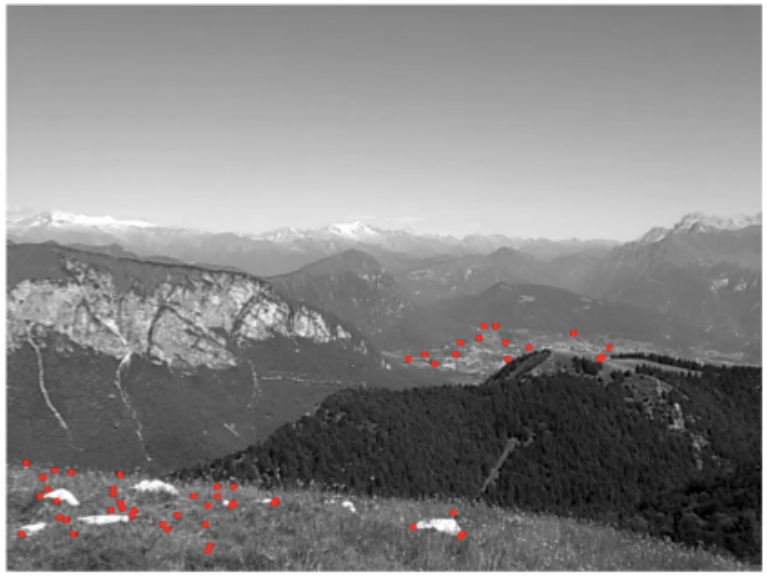




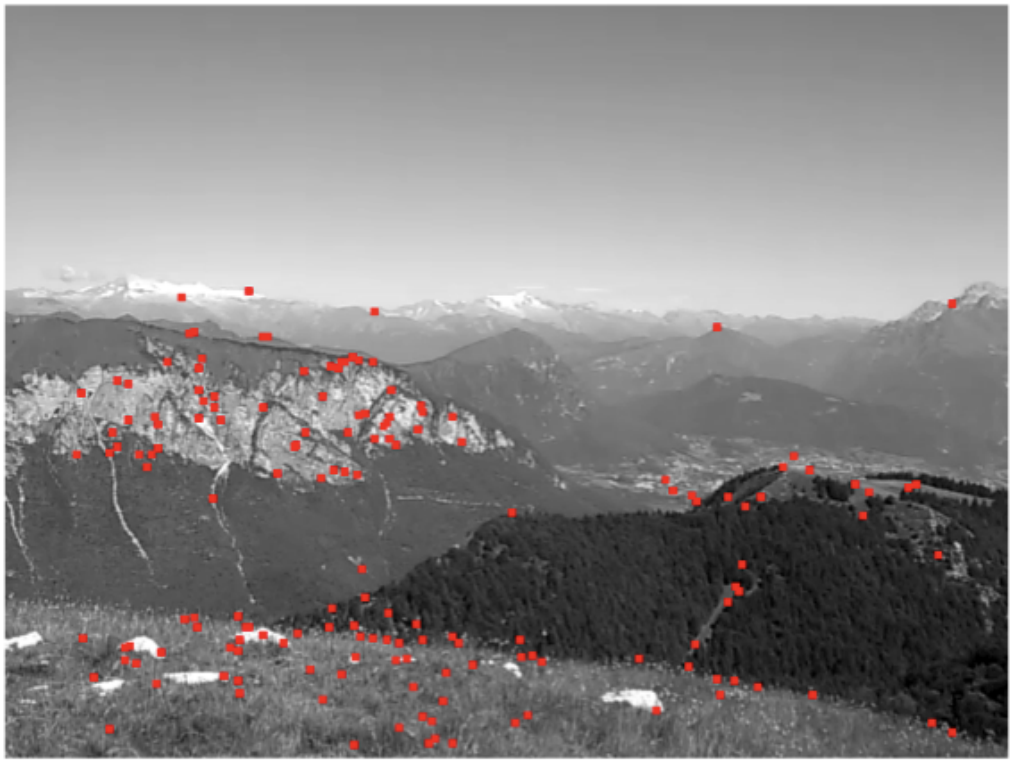


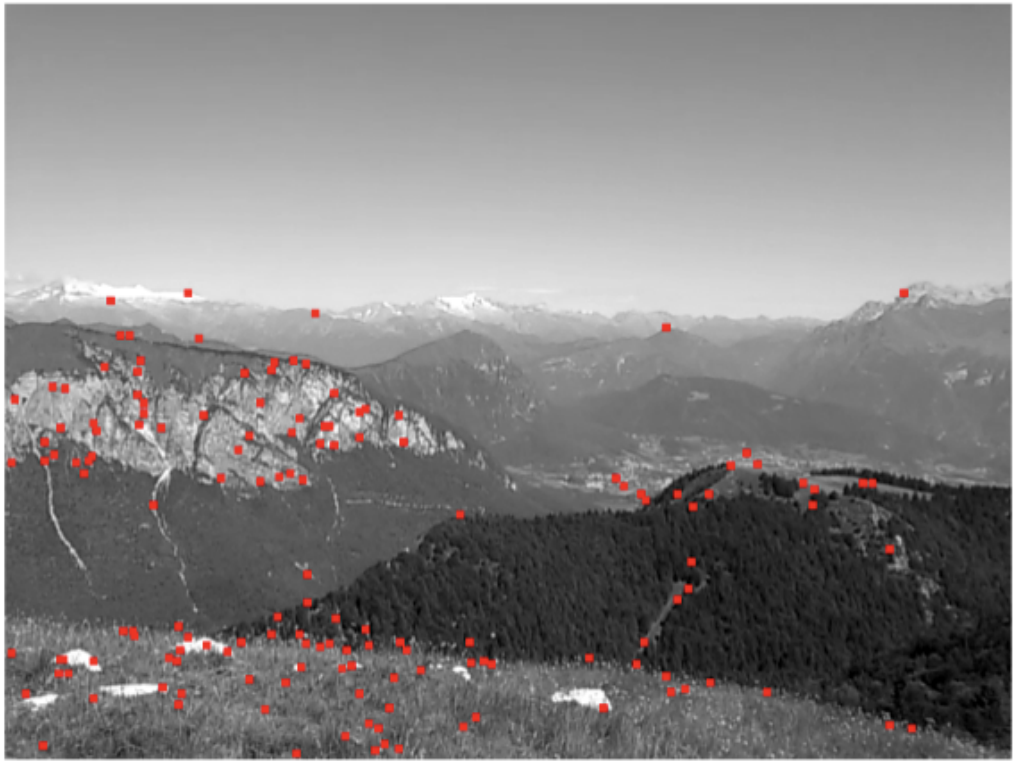
corners of blobs in kmpp segmentation.





matched corners from euclidean solution





matched corners from canny edge corners found with
blob calculated euclidean solution

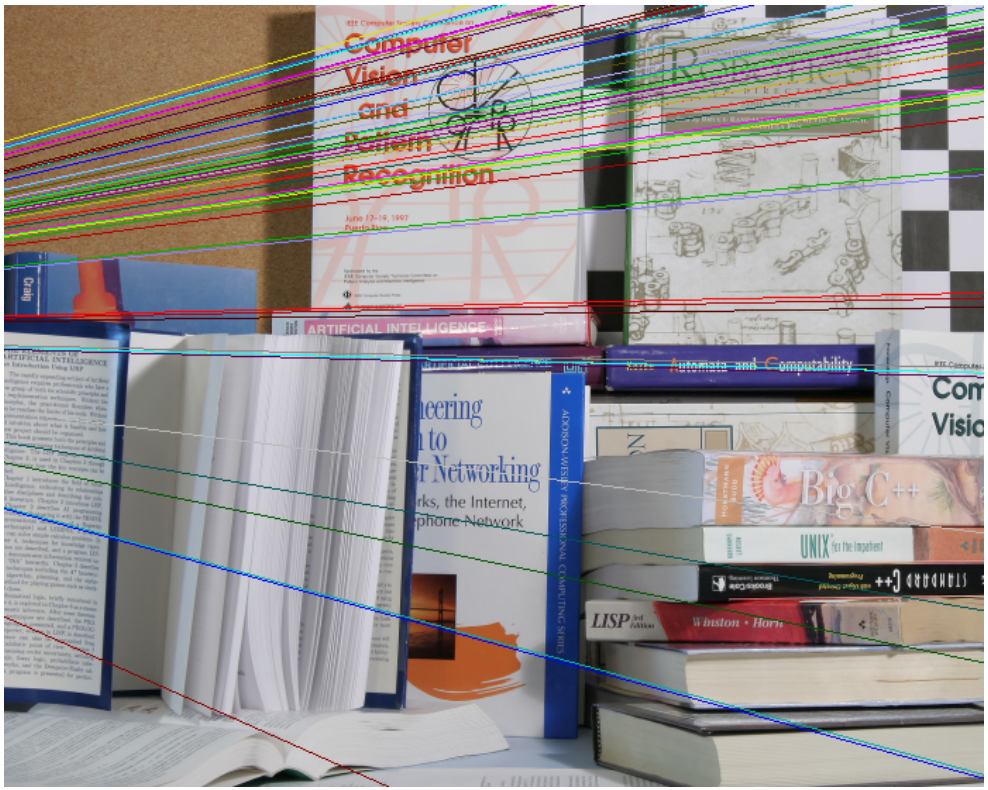
(before RANSAC and epipolar solver)

corners of blobs in kmpp segmentation.

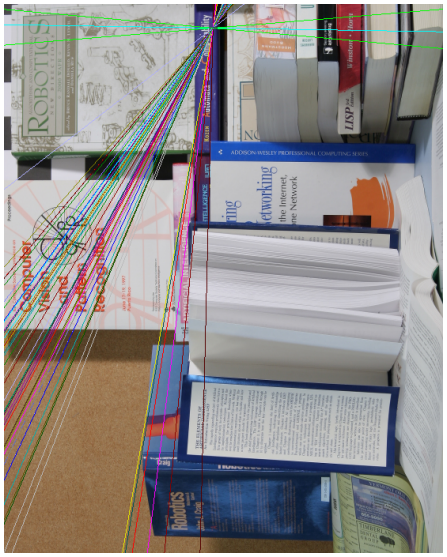
matched corners from euclidean solution

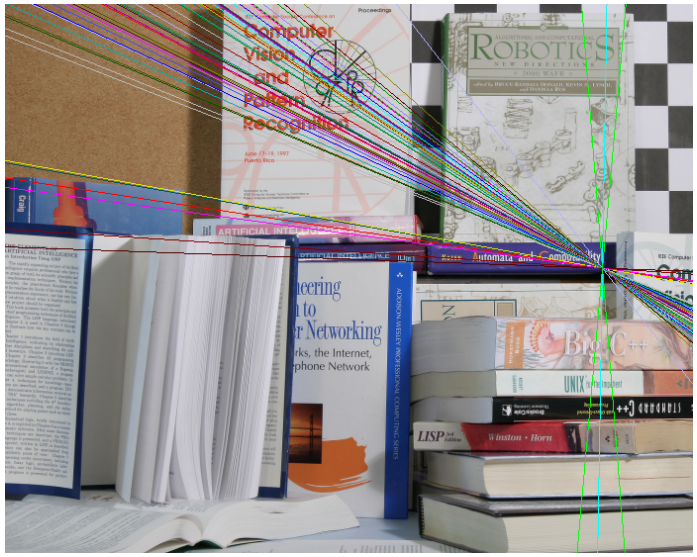
matched corners from canny edge corners found
with blob calculated euclidean solution

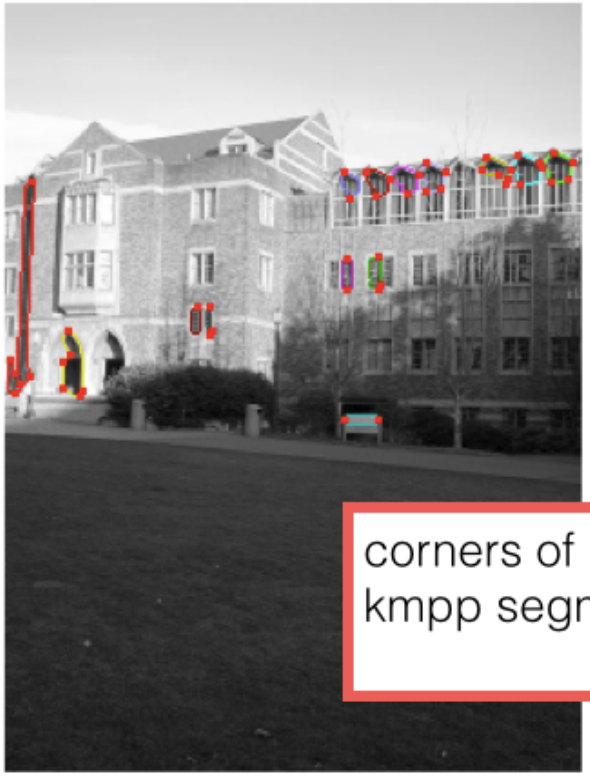
(before RANSAC and epipolar solver)

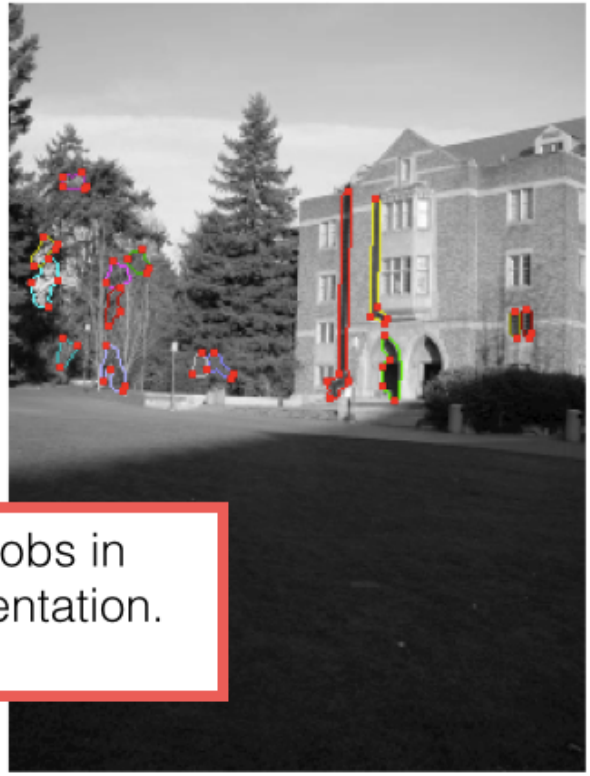


needs improved corners (blobs, segmentation)







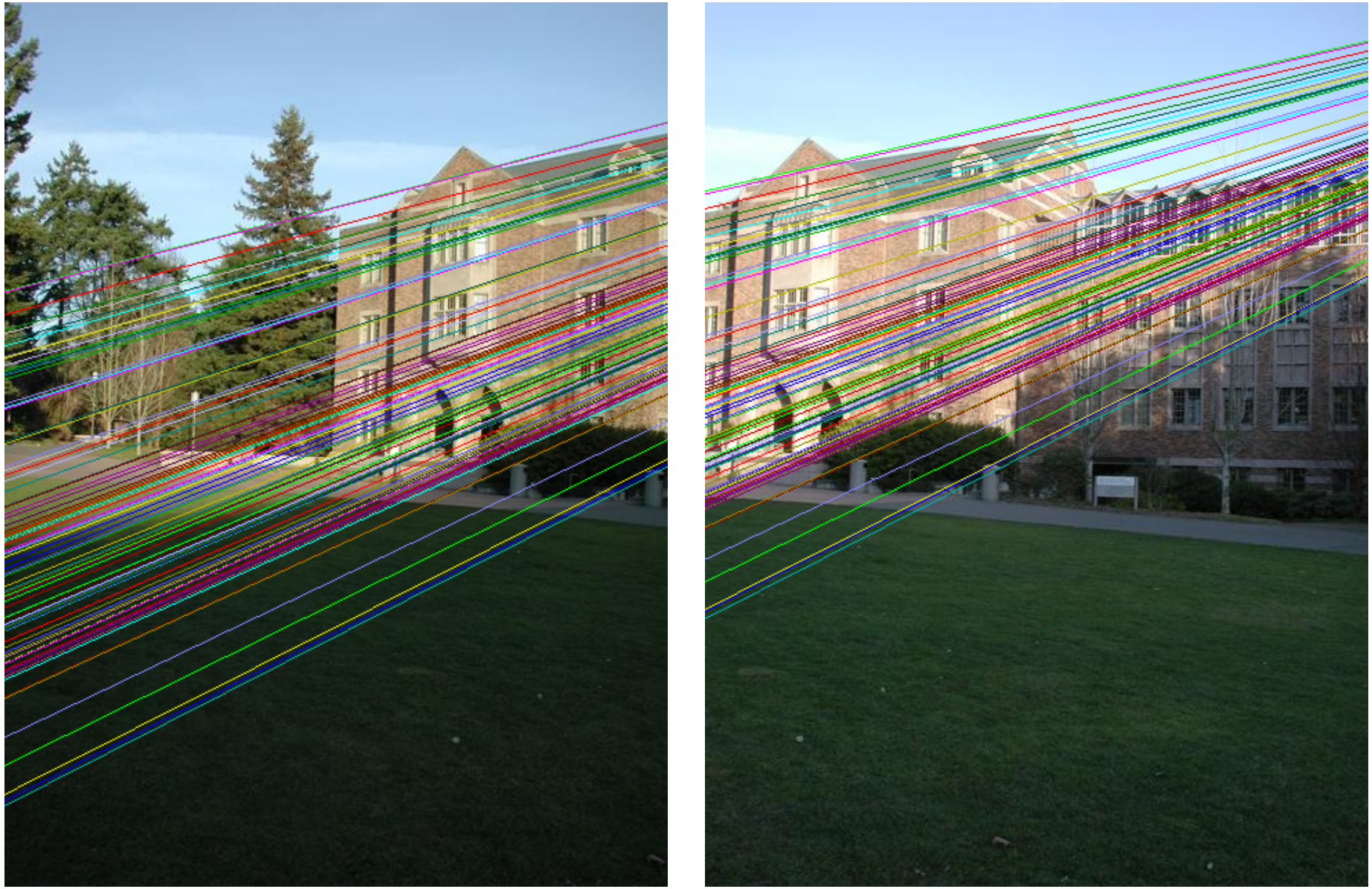


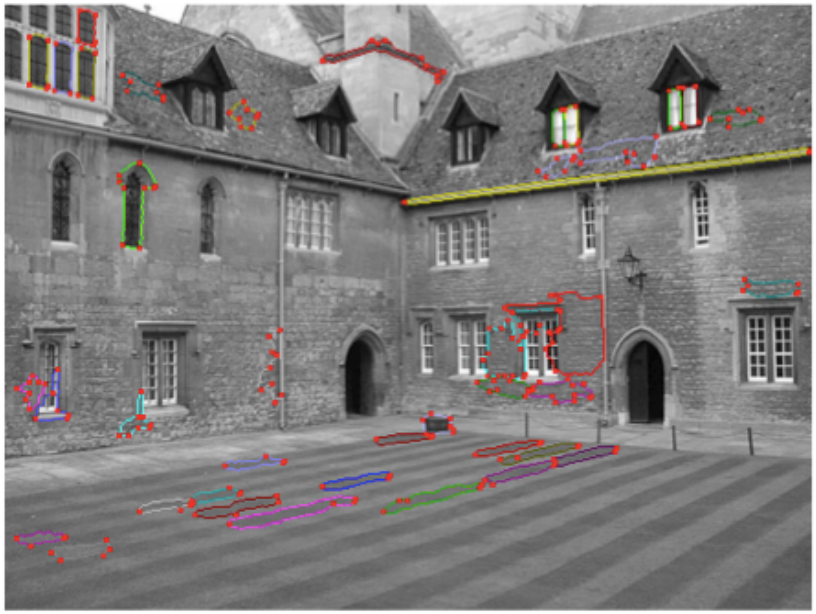
corners of blobs in kmpp segmentation.

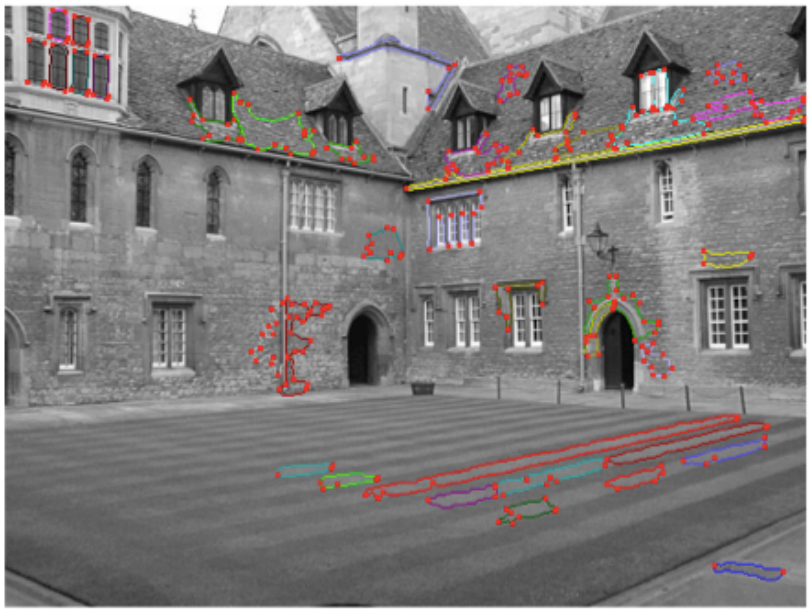
matched corners from euclidean solution

matched corners from canny edge corners found
with blob calculated euclidean solution

(before RANSAC and epipolar solver)





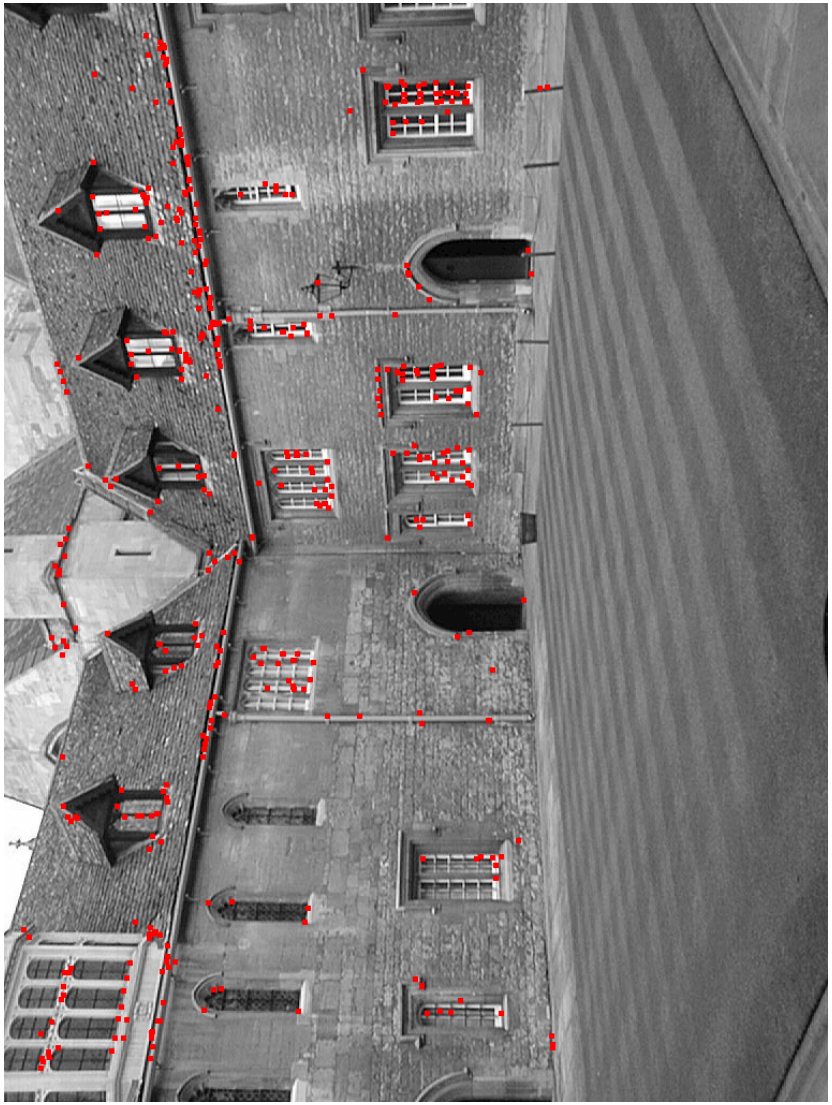


corners of blobs in kmpp segmentation.

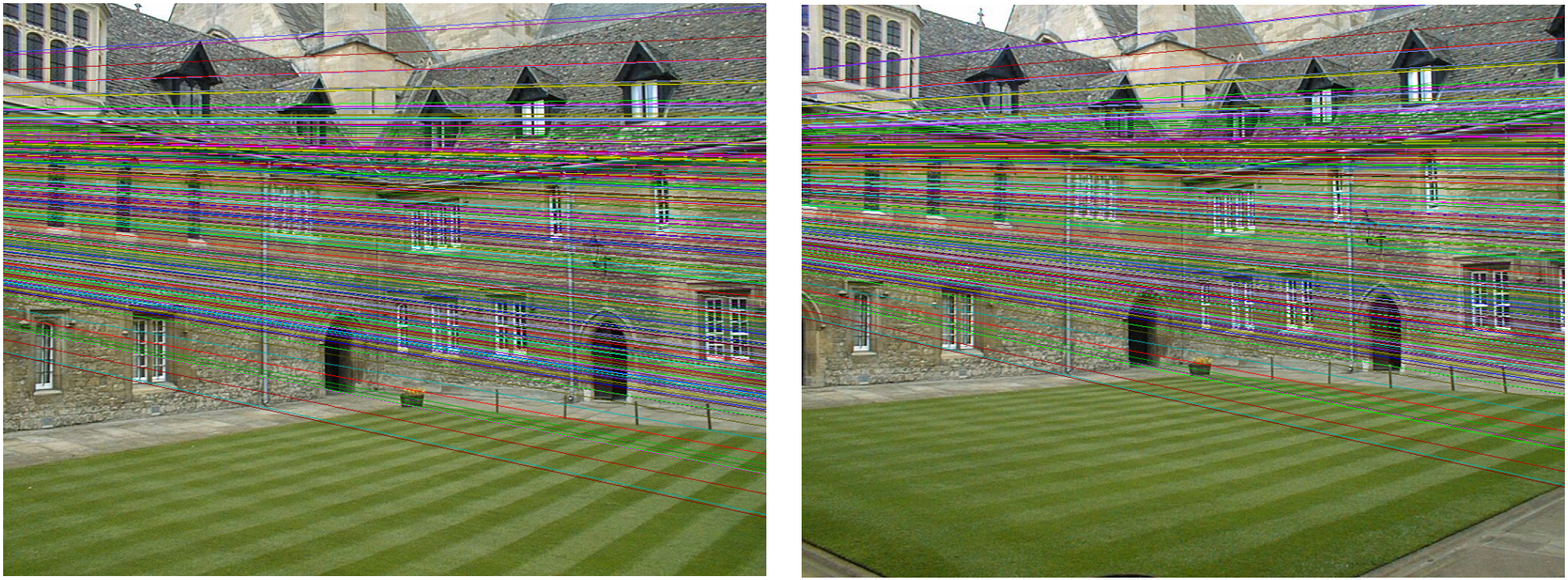
matched corners from euclidean solution

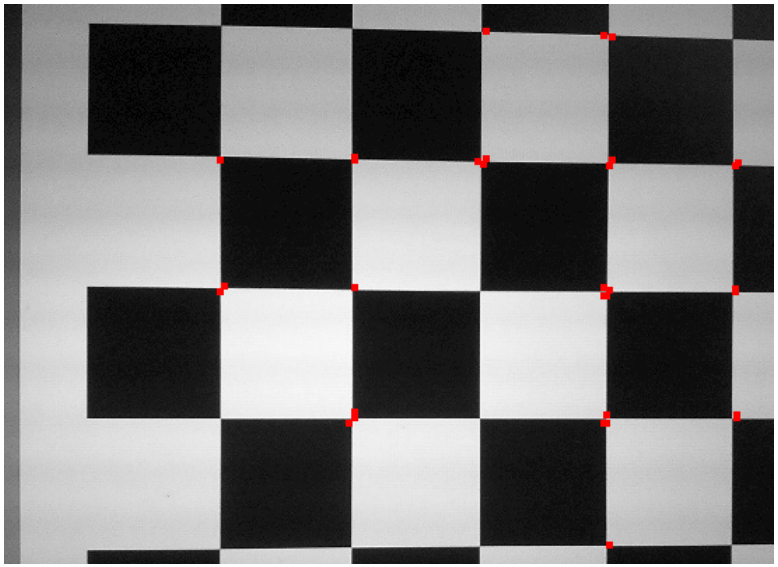
matched corners from canny edge corners found with blob calculated euclidean solution.

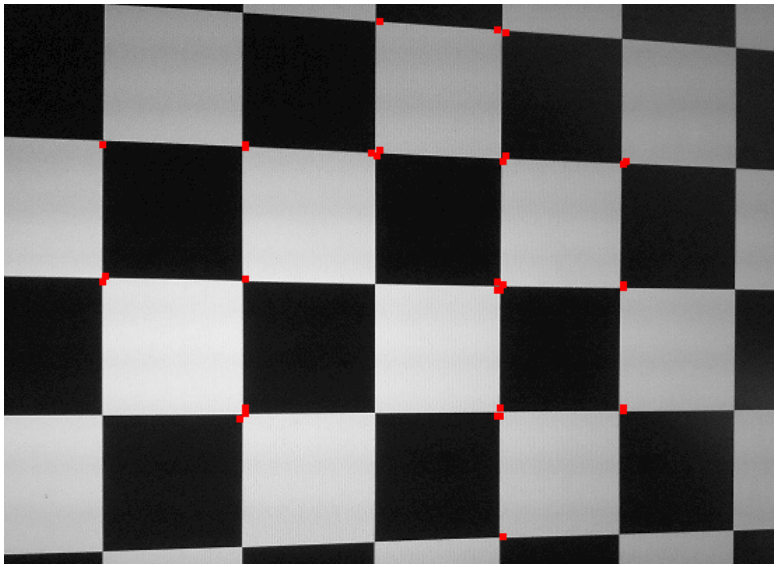
(before RANSAC and epipolar solver)



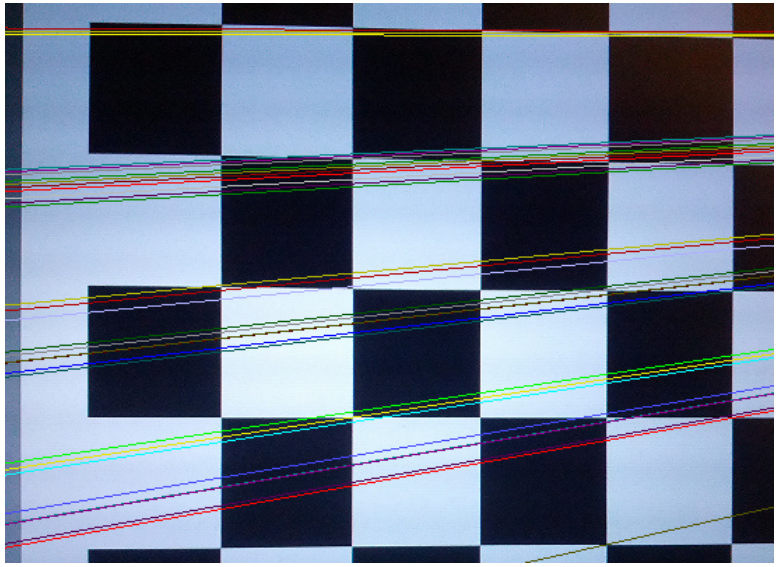
same steps, but image1 is landscape

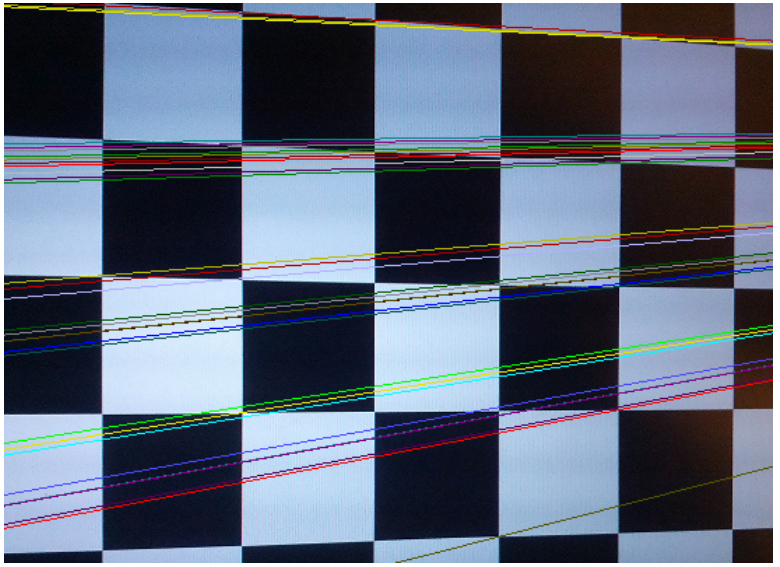




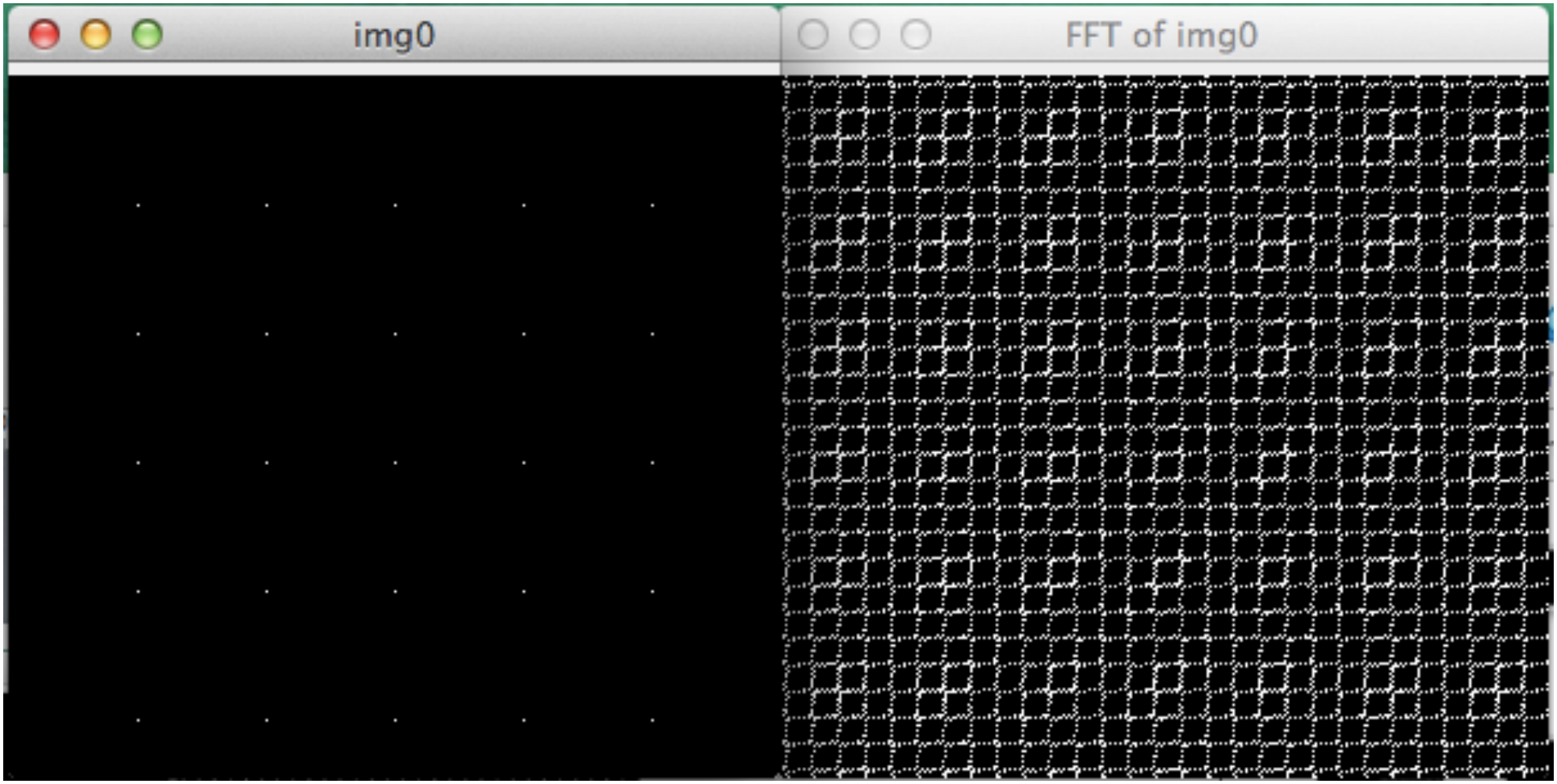


matched corners

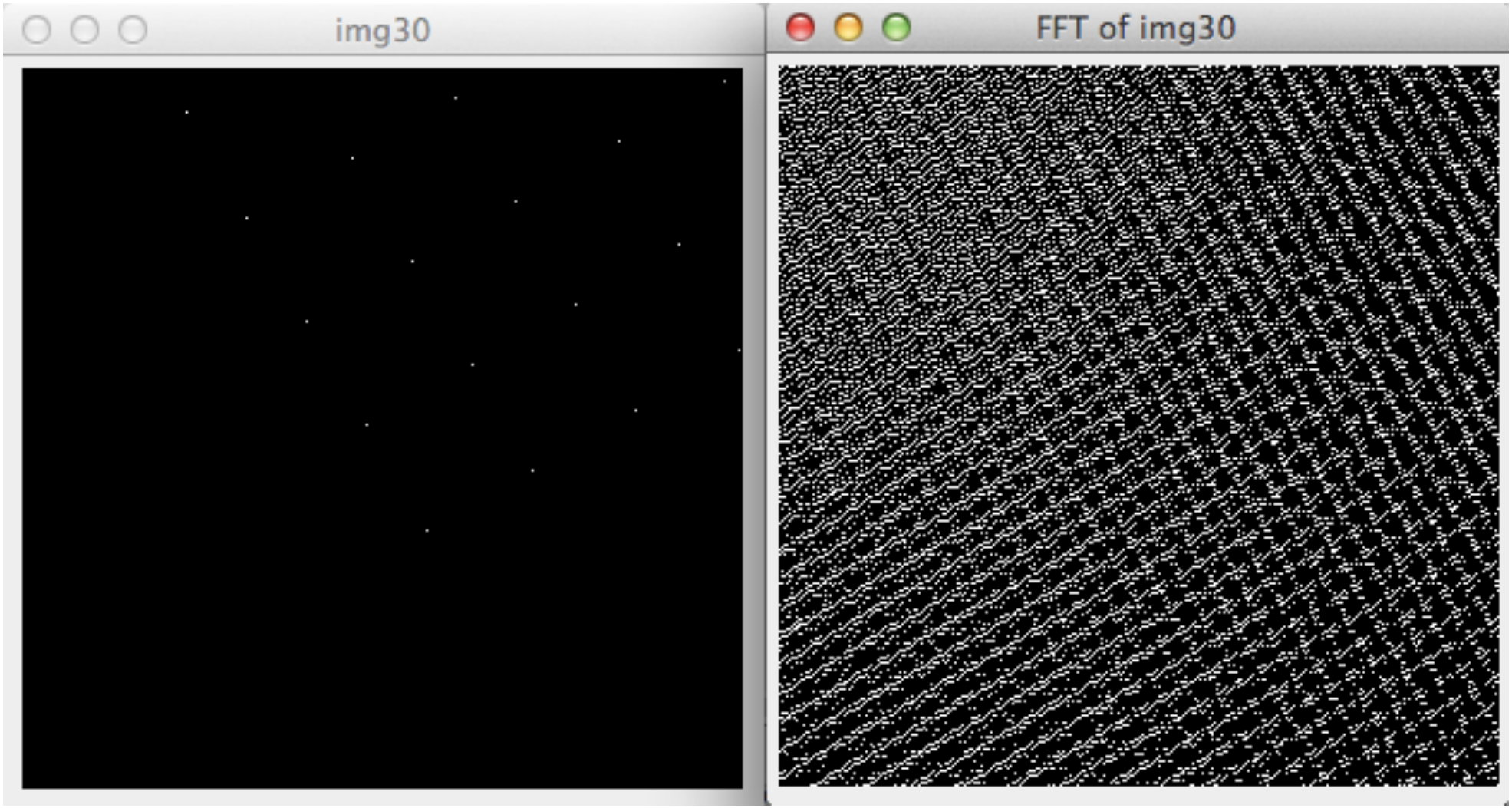




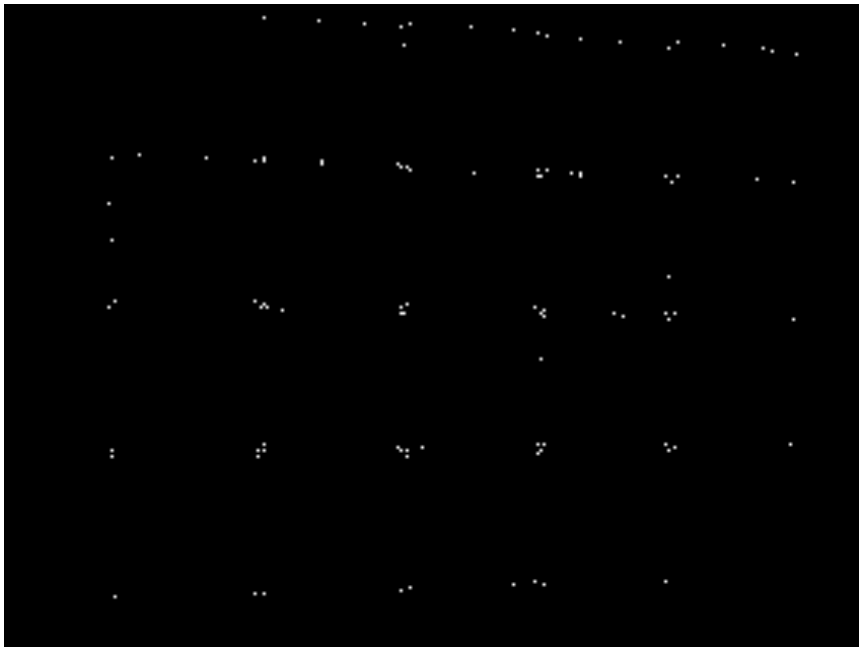
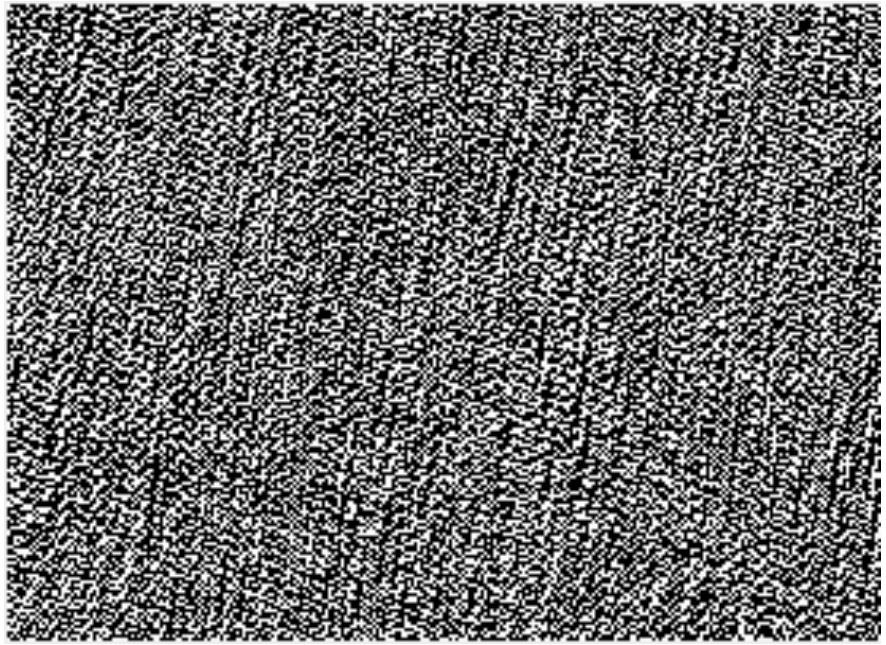
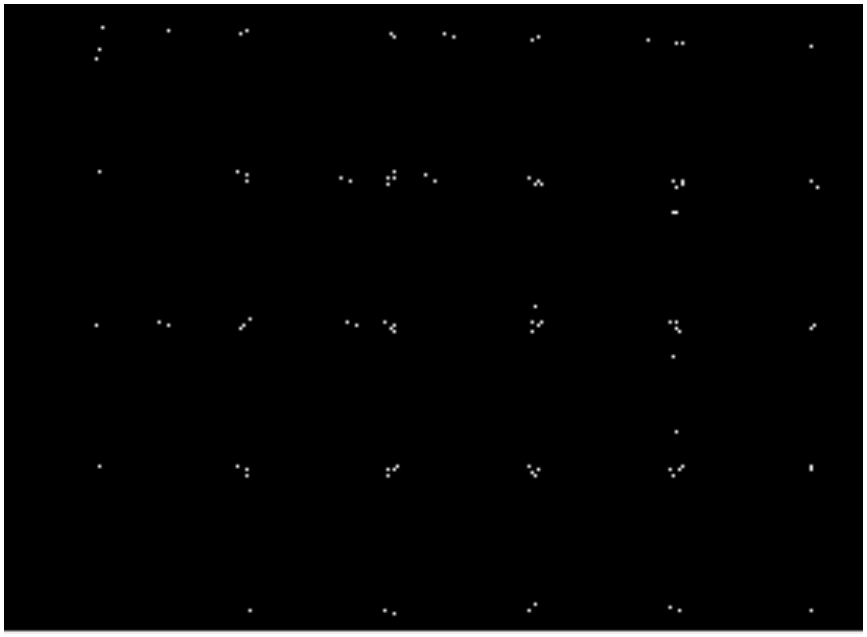
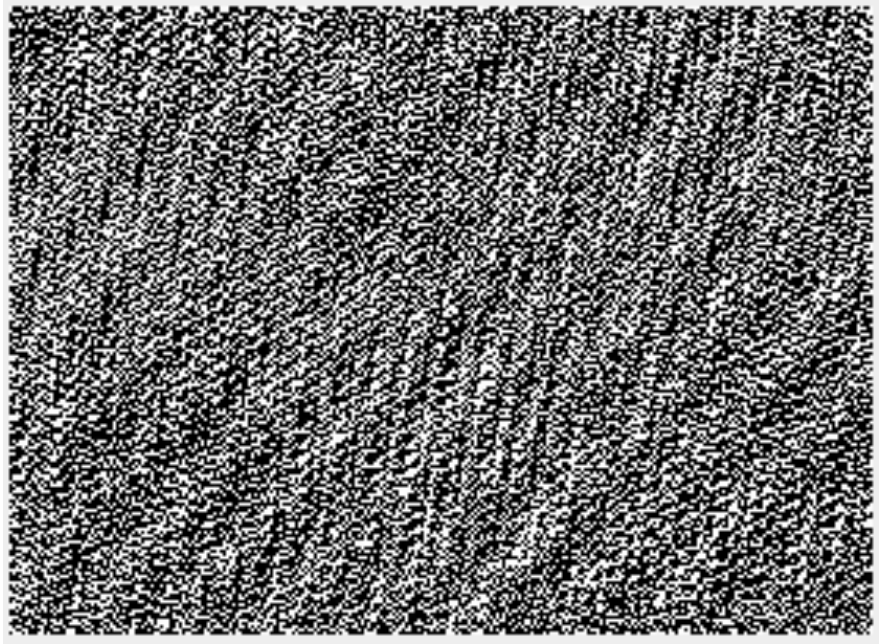
epipolar projection lines



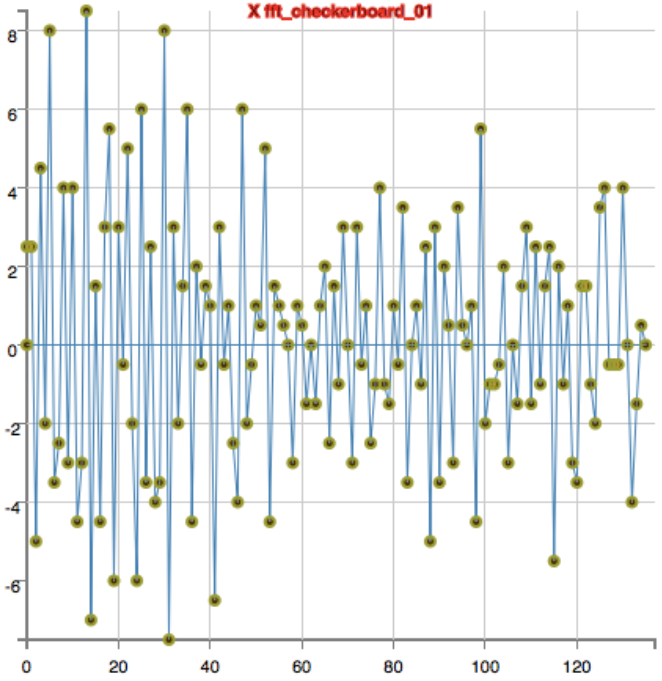
FFT of a grid of points

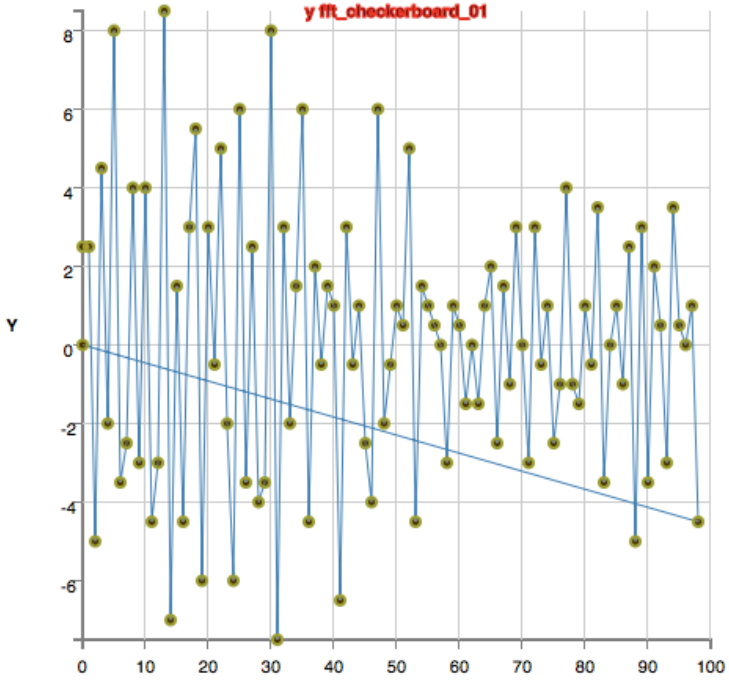


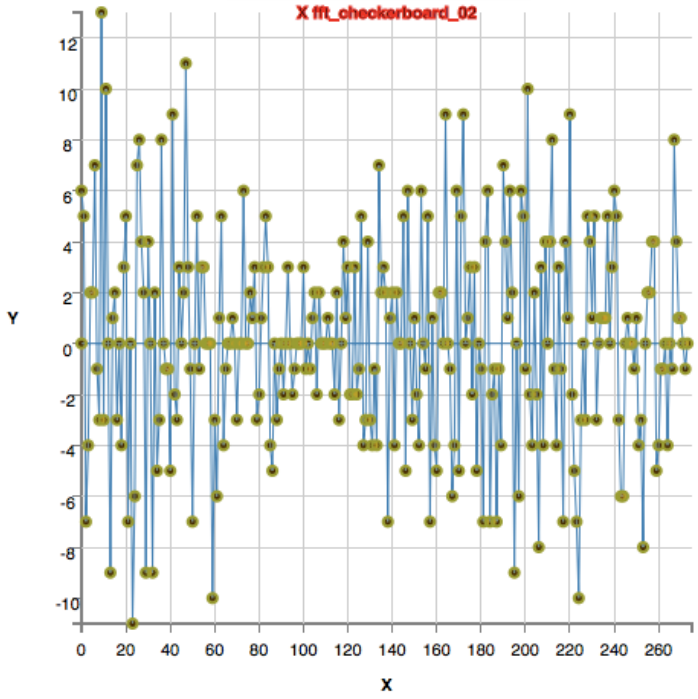
FFT of a same grid of points rotated by 30 degrees

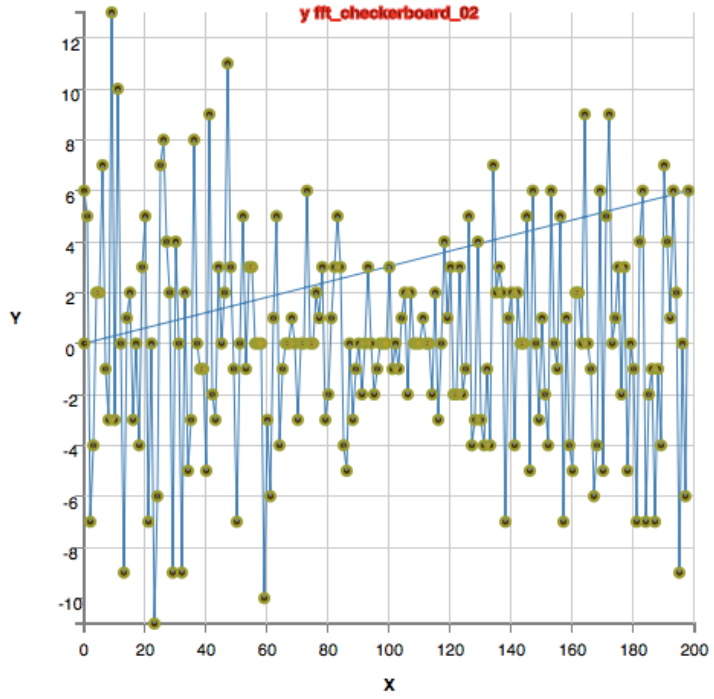





FFT of the corner regions

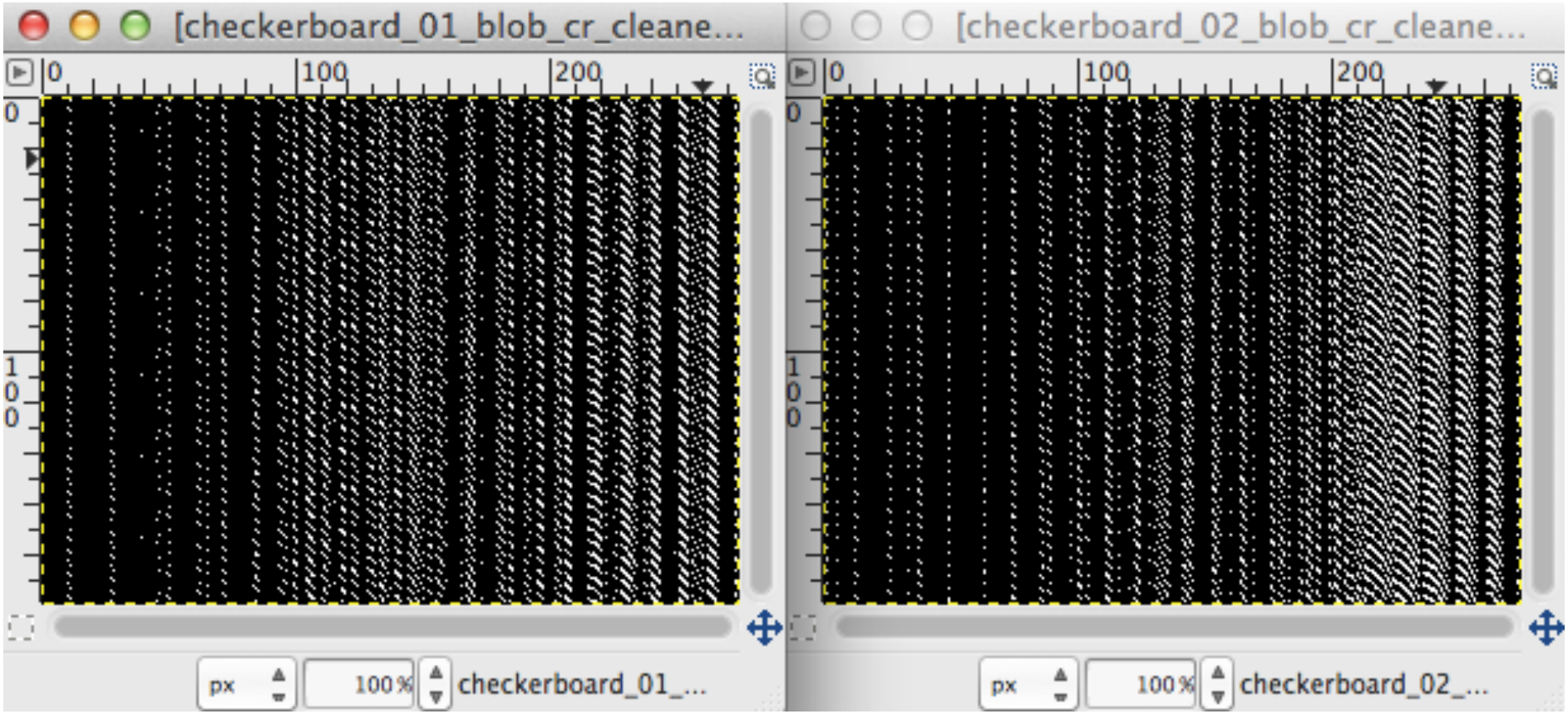




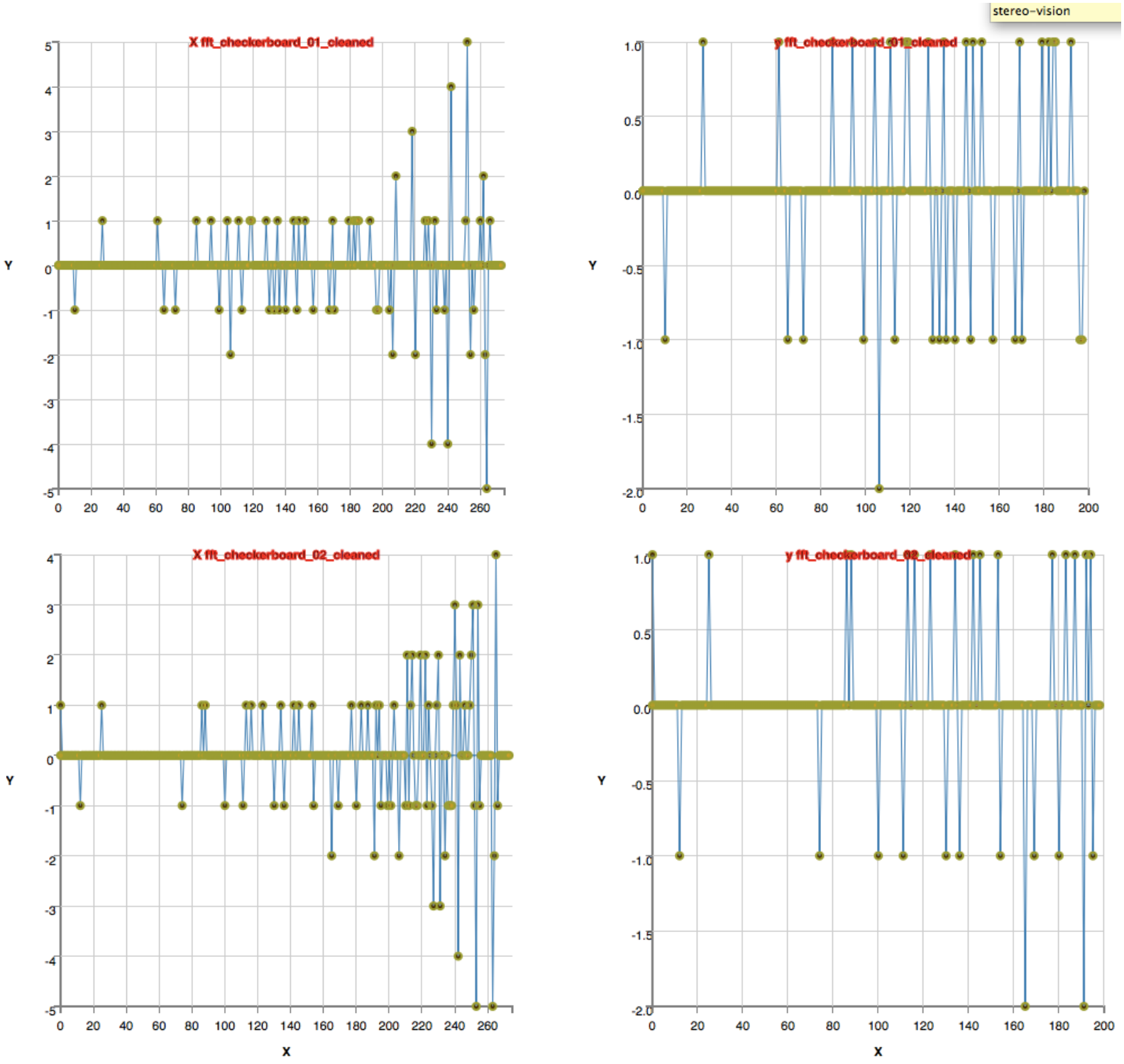




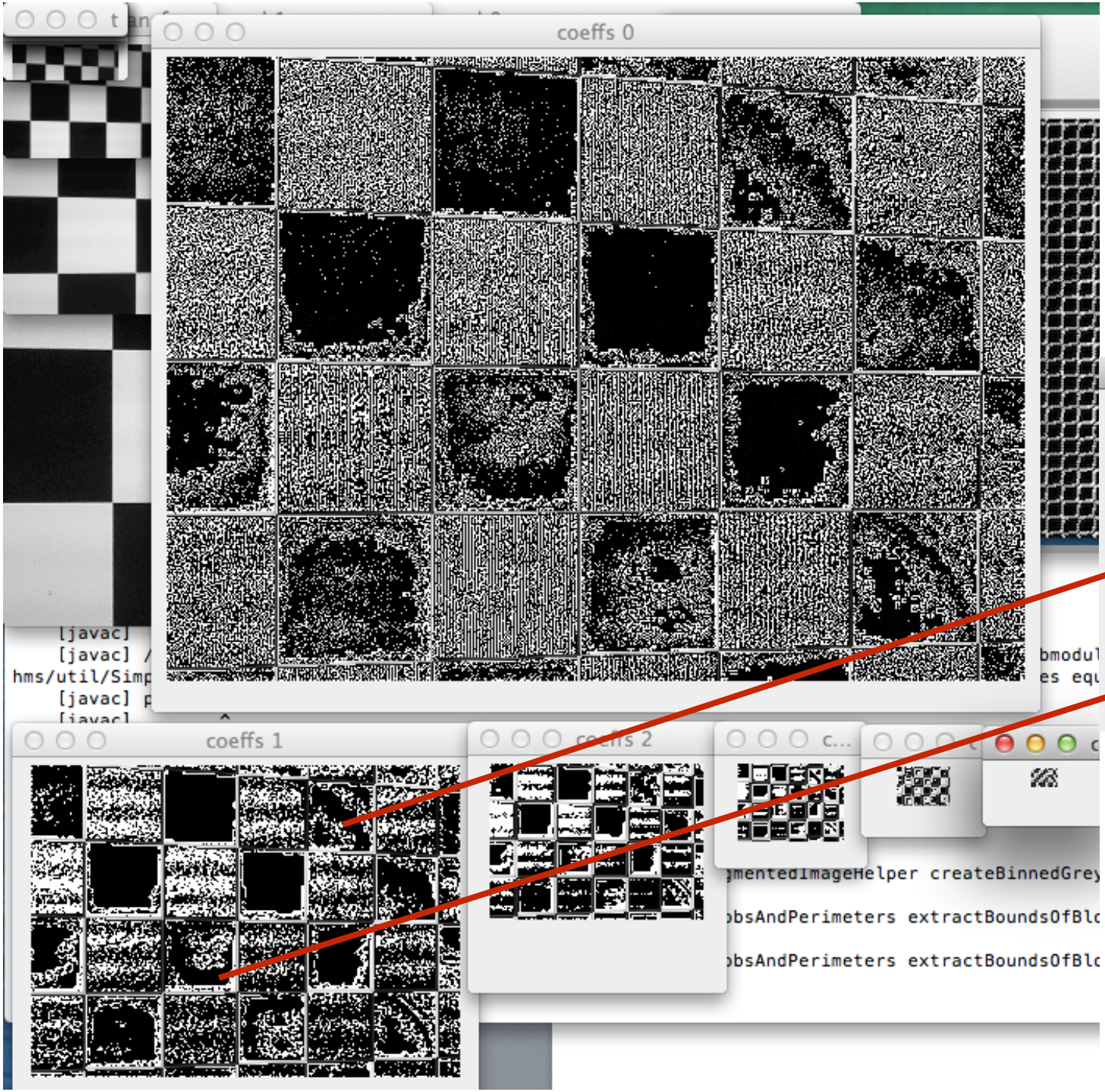
FFT of the corner regions



FFT of corner regions (artifact corners removed)



FFT of corner regions (artifacts removed)

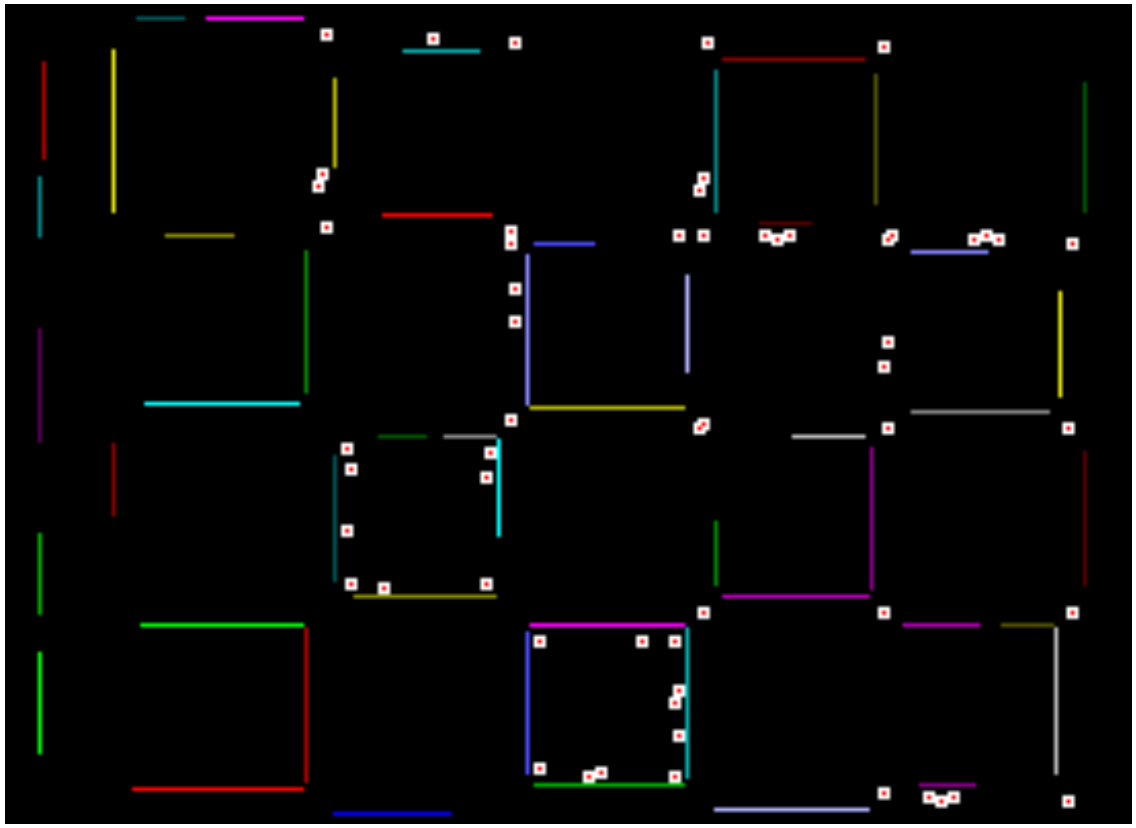


a look at
median transform
coefficents
for potential use
in segmentation

The a trous wavelet transform finest coefficient is now what the blob segmentation is based upon.

The checkerboard test image is an example that results in corners from line artifacts and those artifact corners need to be removed for feature matching between images.

The Hough transform for lines looks useful in finding straight line segments of a minimum length and using a subsequent fit to approximate line segment endpoints that can be used to remove points in between endpoints which are not part of real junctions.



corners from the a trous segmentation over plotted on the lines found from Hough Transform for lines w/ a small theta and small radius tolerance.

Can see that it should be possible to remove the artifact corners.