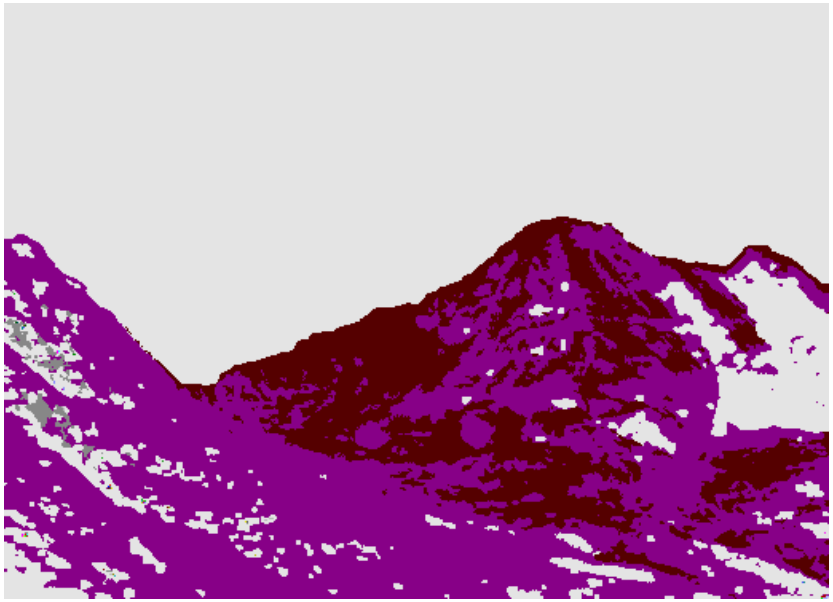
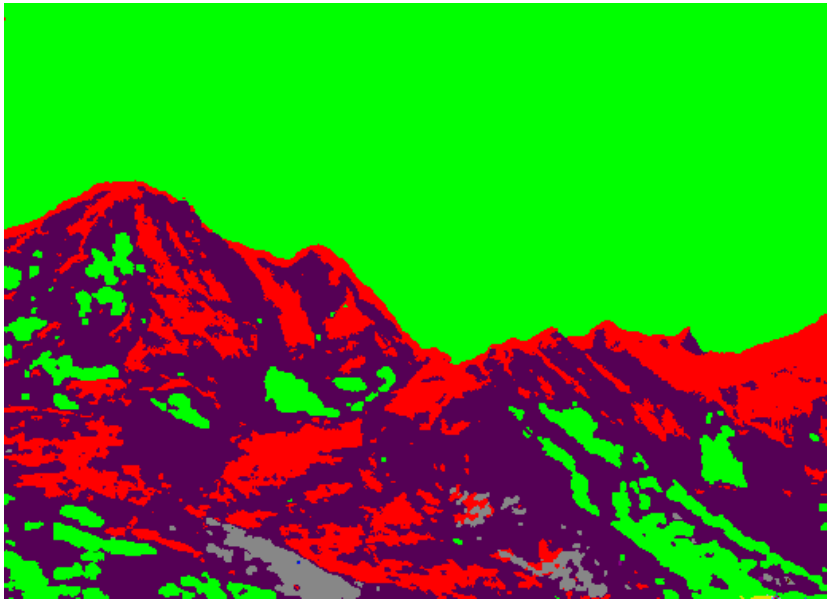
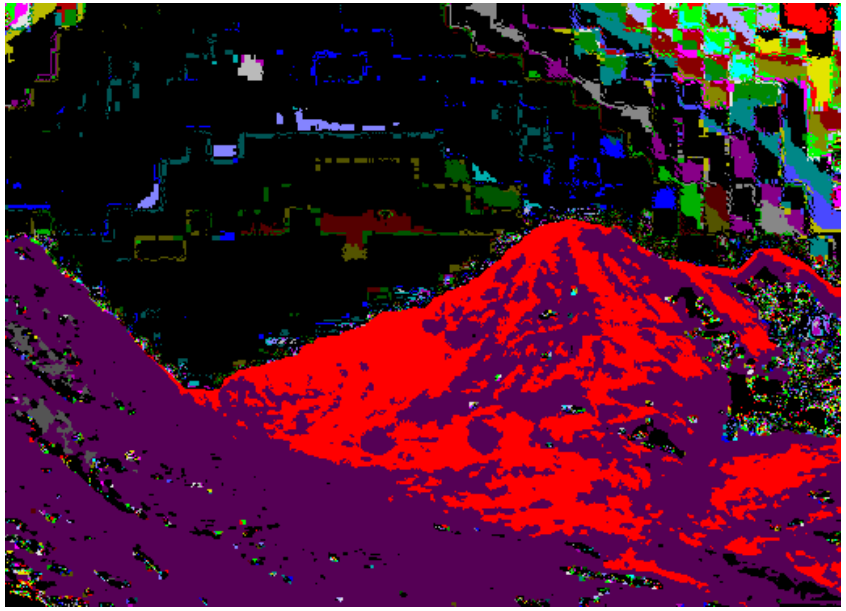
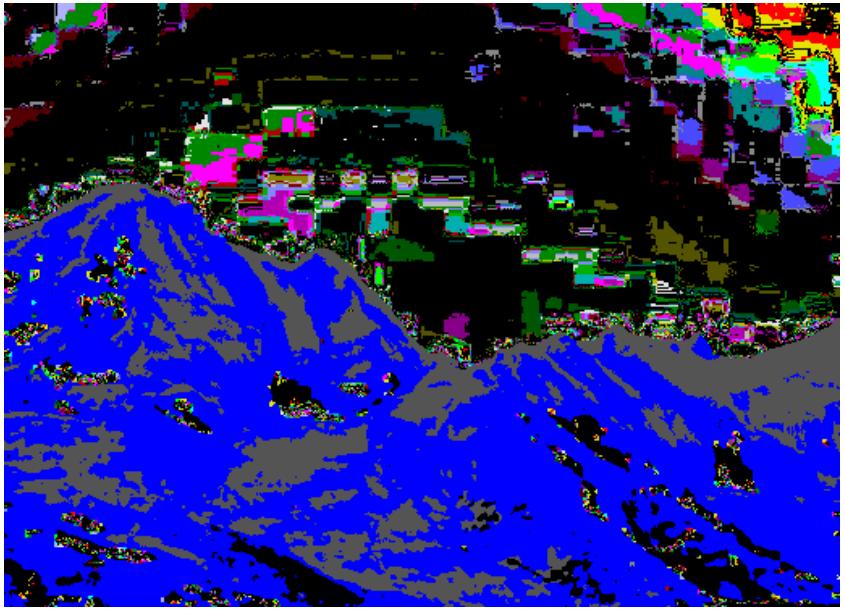
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);



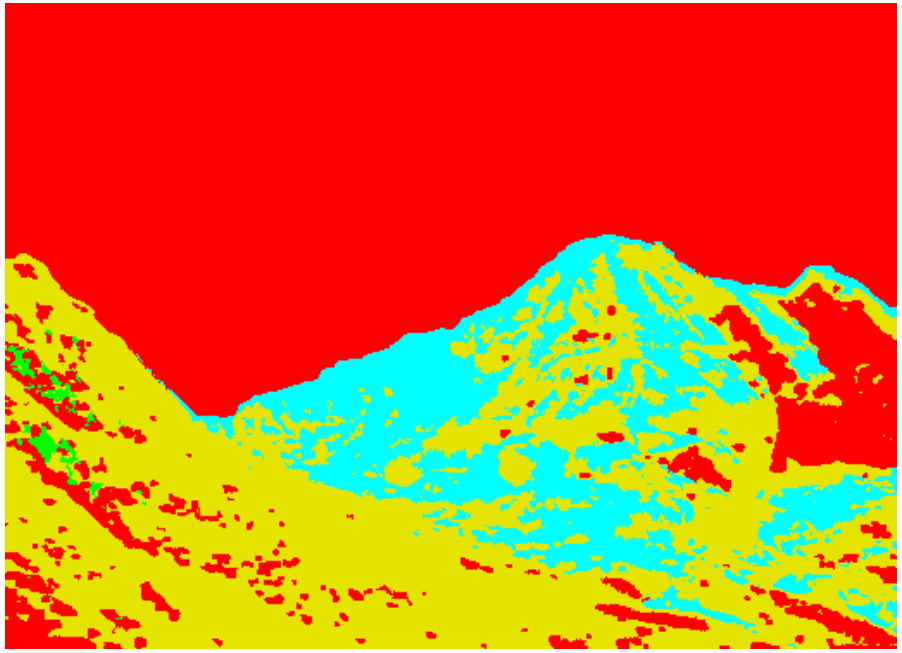


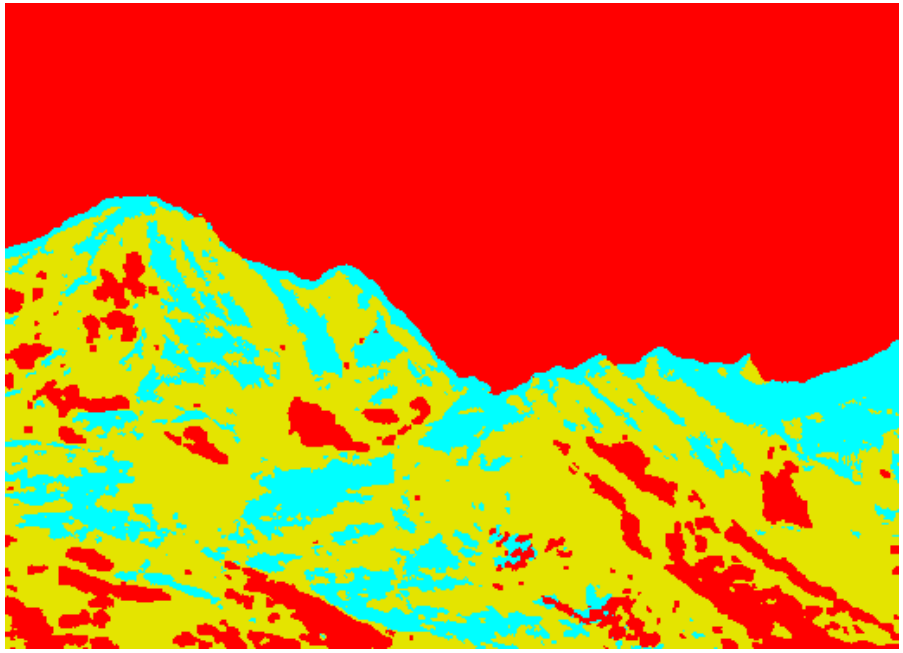
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);



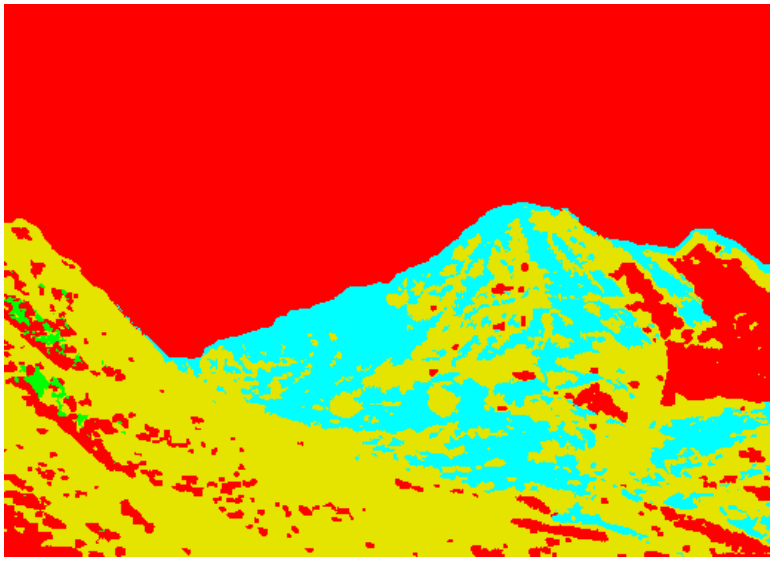


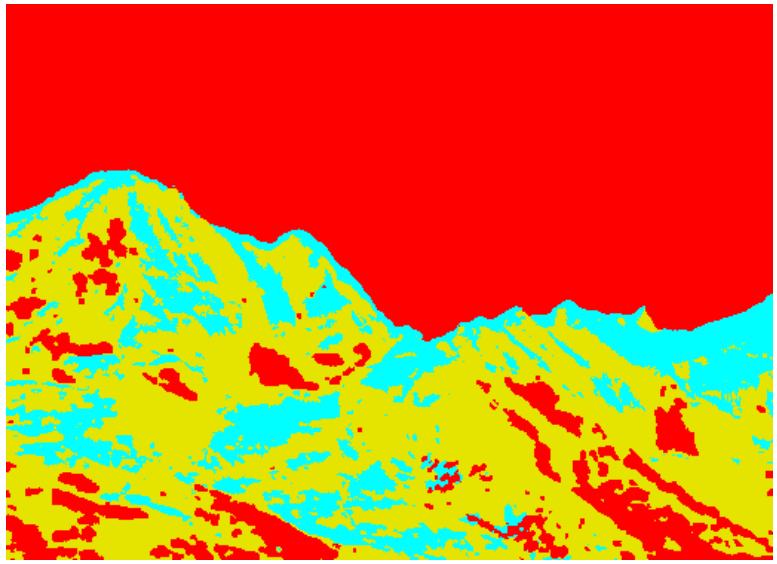
```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);
```





```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);
```





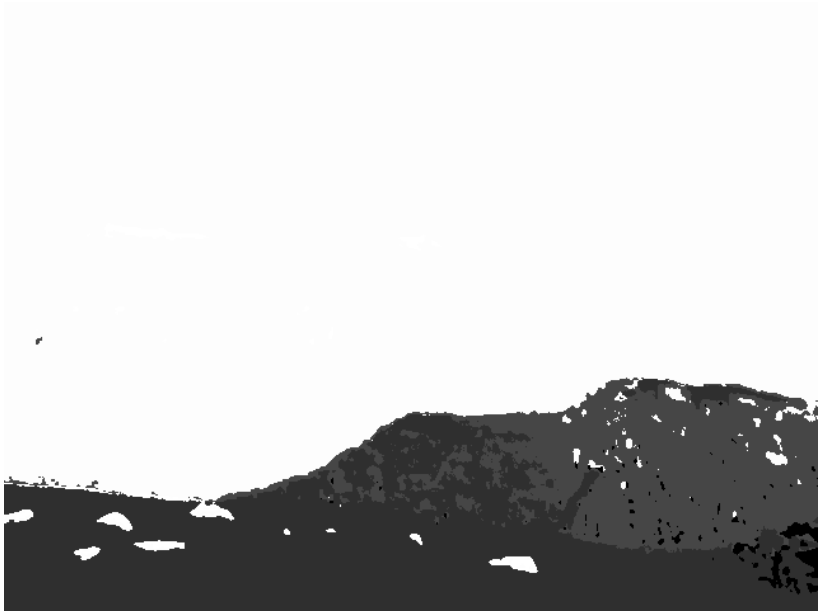
```
int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

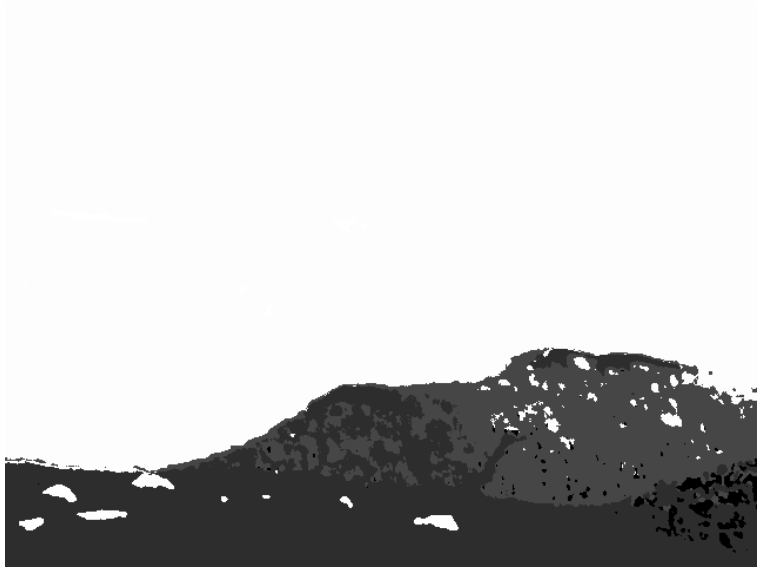
```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);

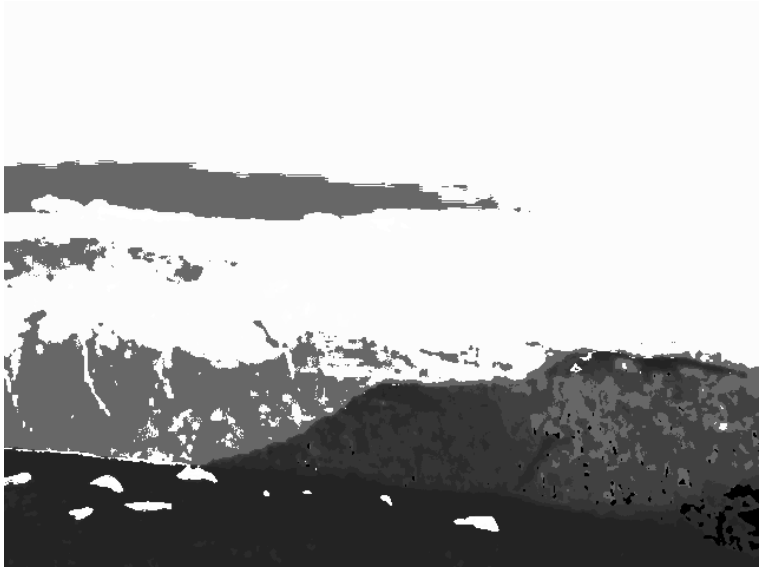
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

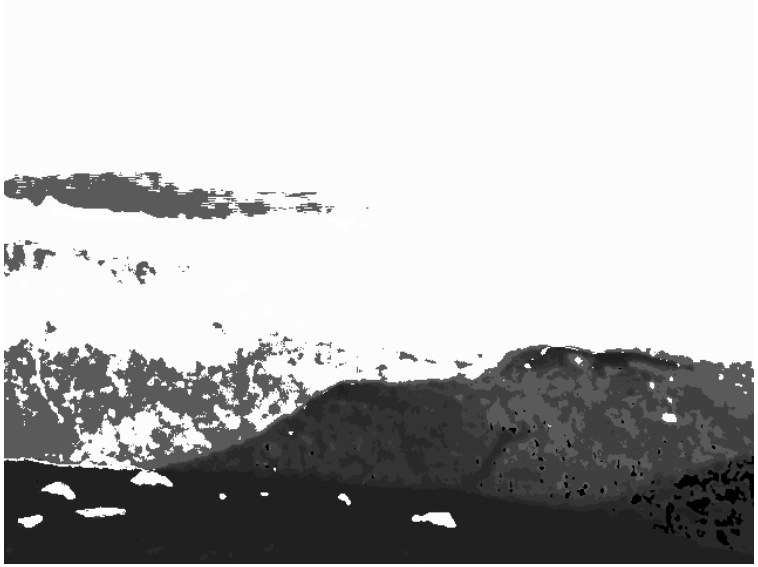
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);





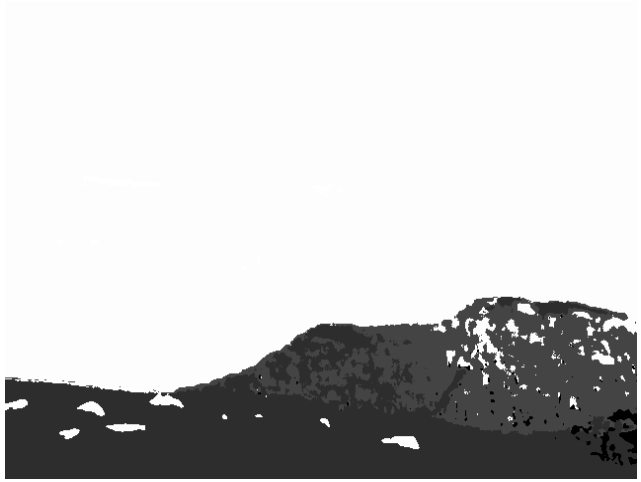
int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

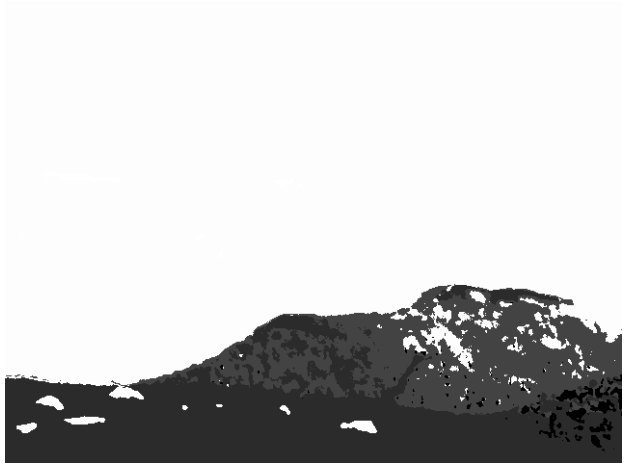




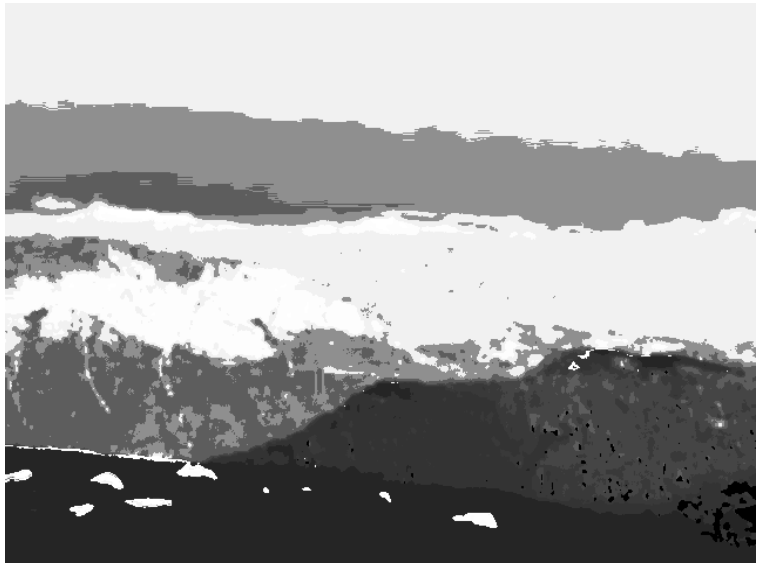
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

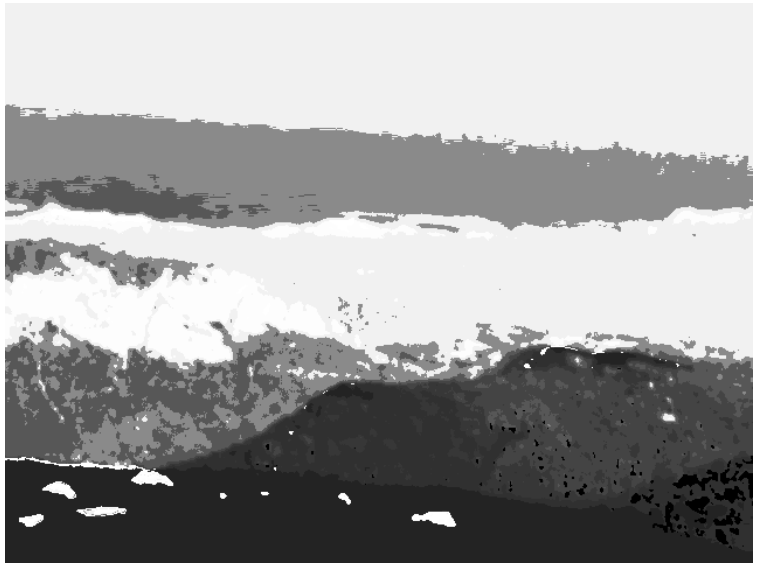
int kBands =3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);





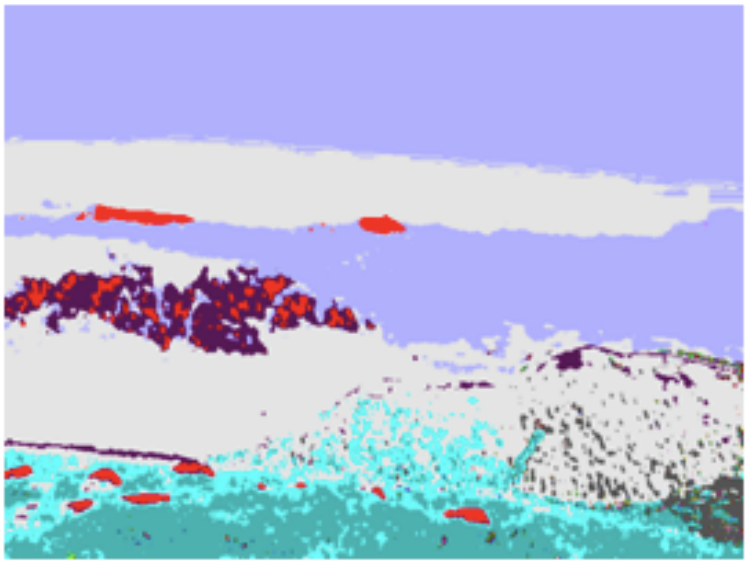
int kBands =8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq

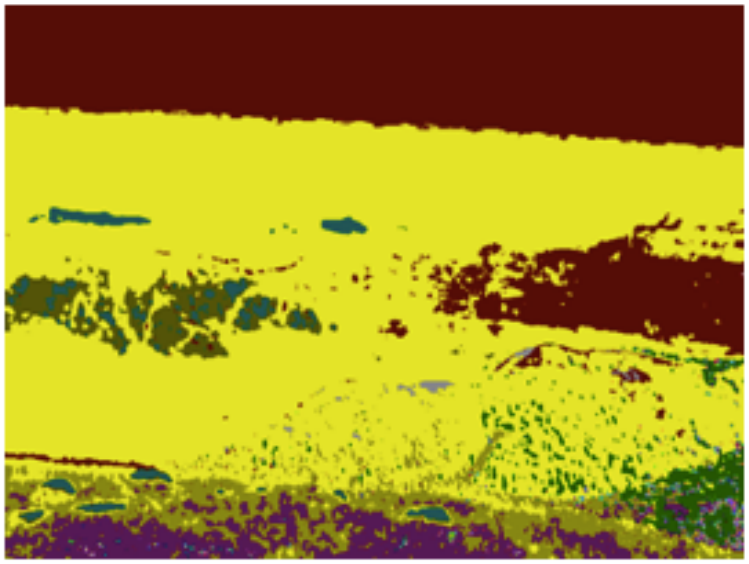




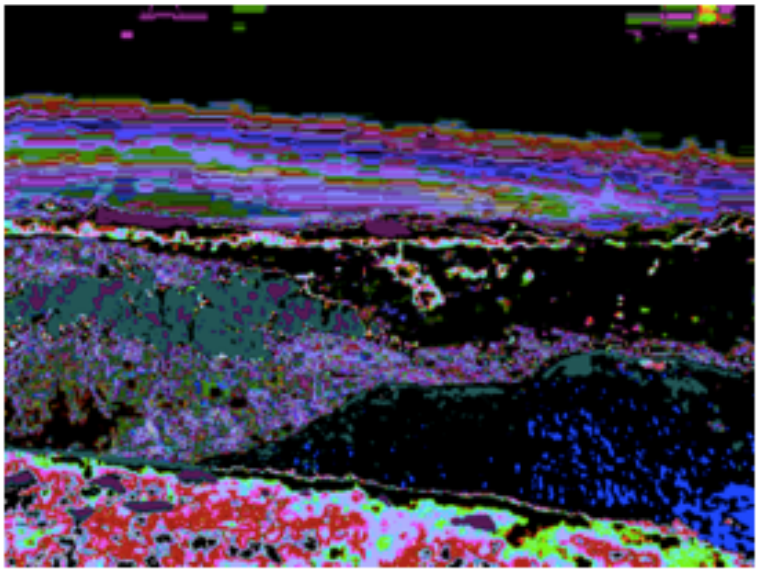
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistogram(gsImg1, kBands);

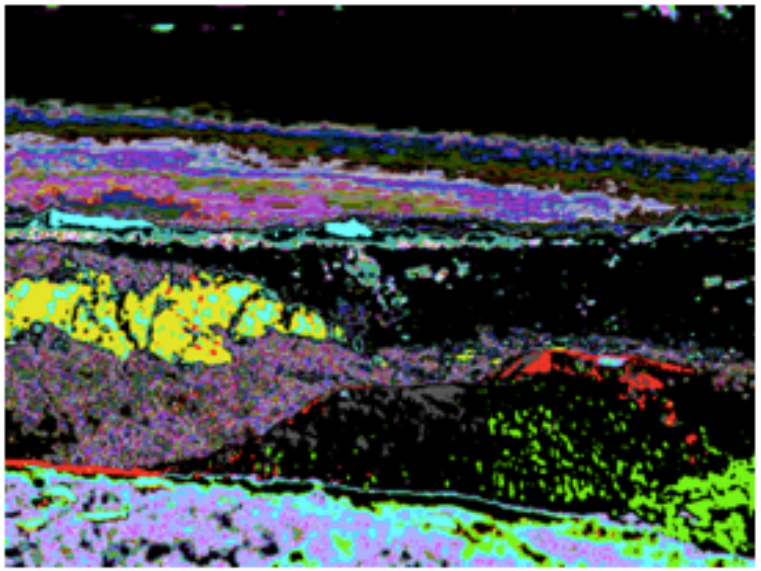
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);



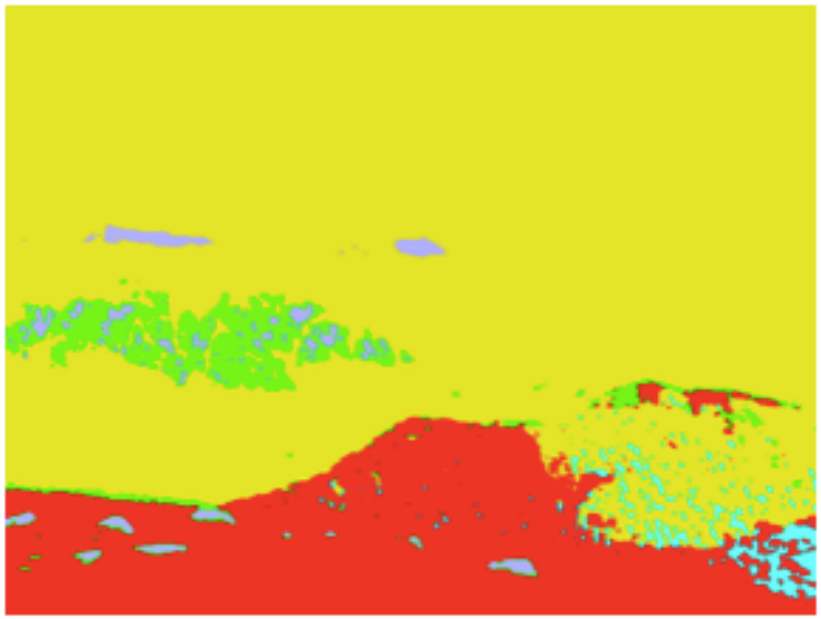


imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);

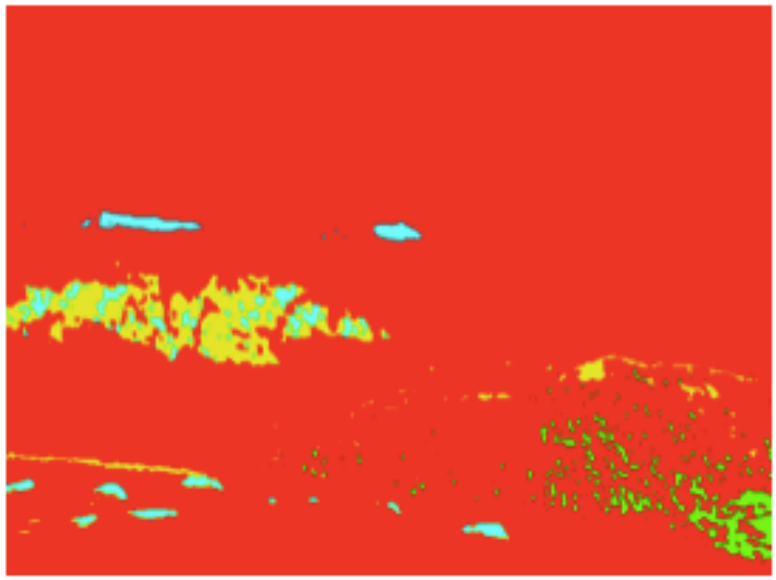


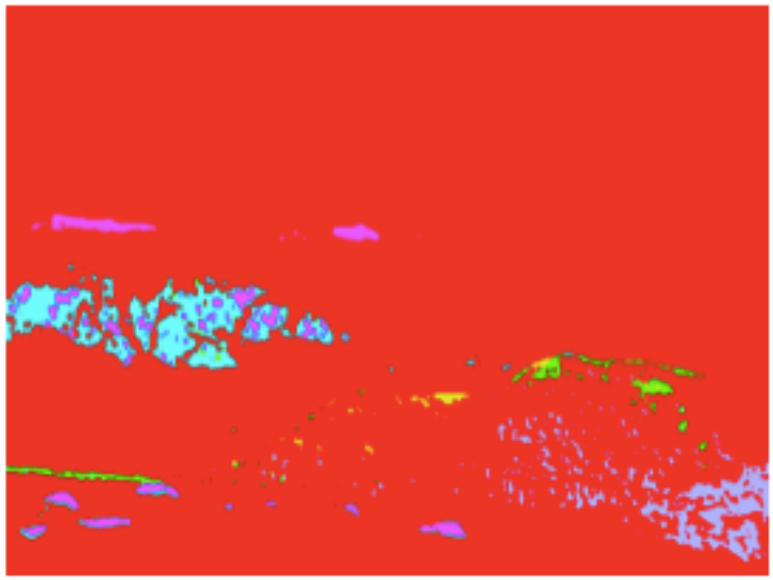


imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);



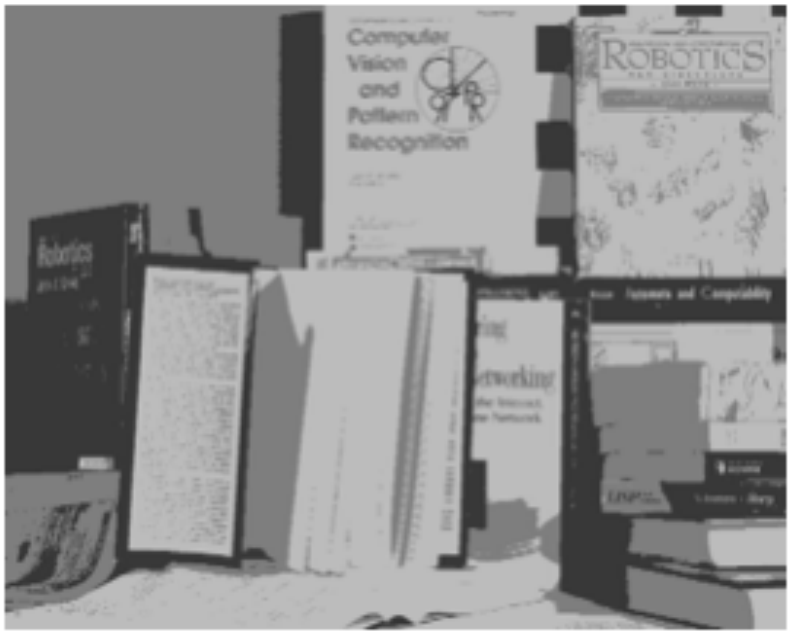
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);

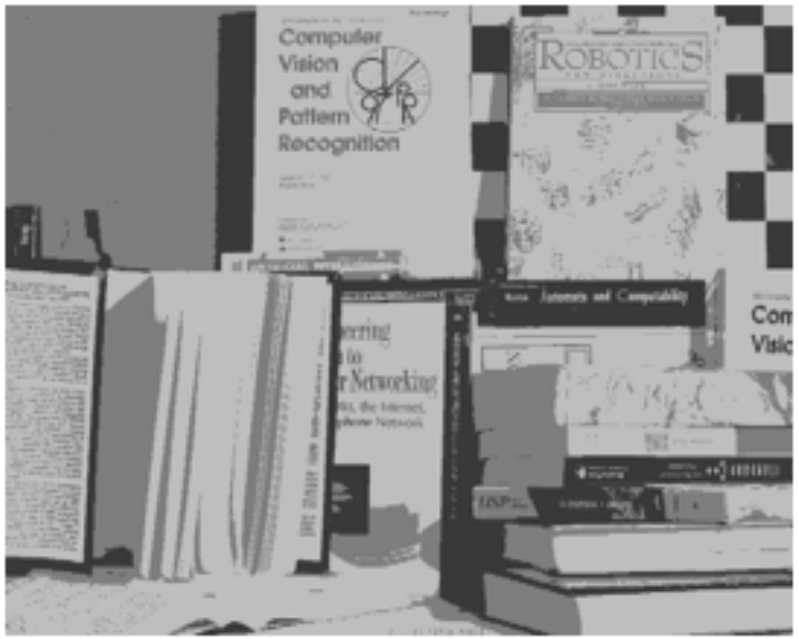




int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);

int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);



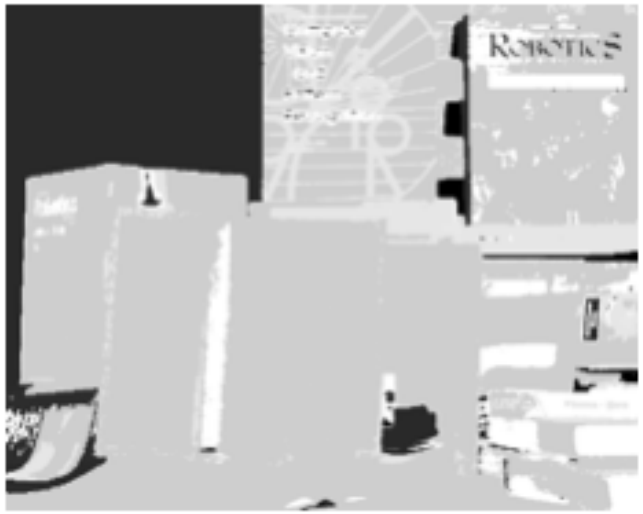


int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);

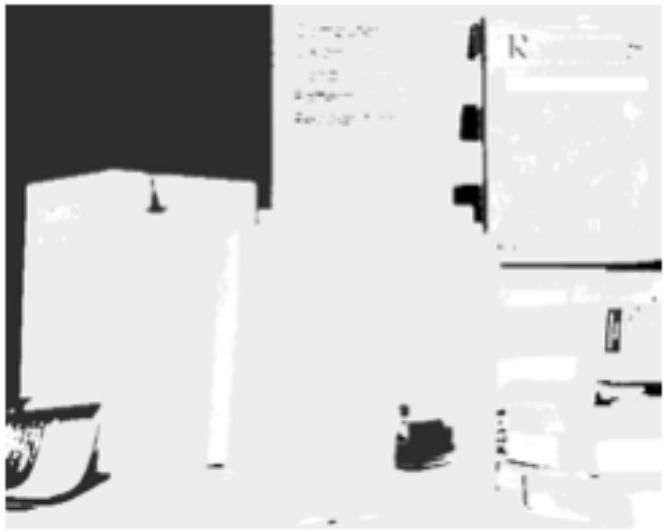
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

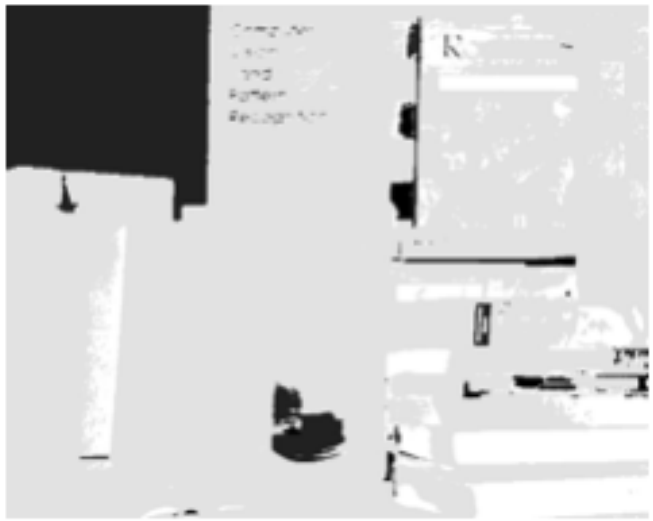
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

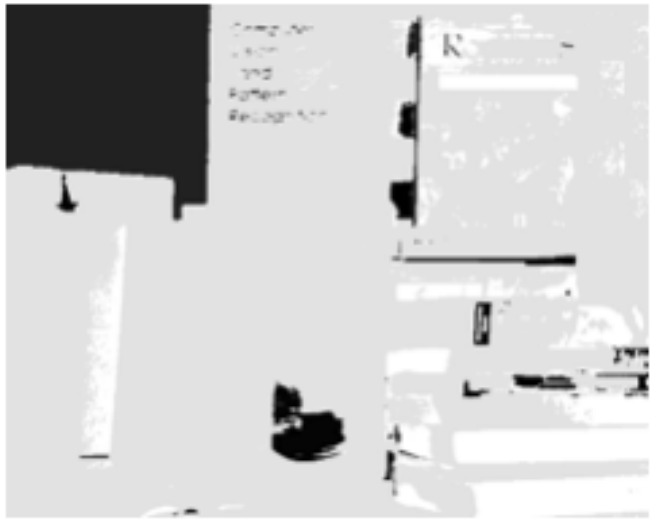


int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

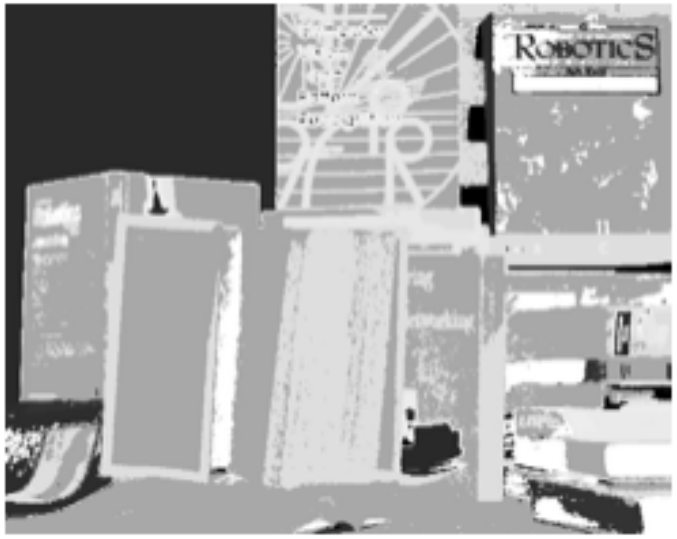


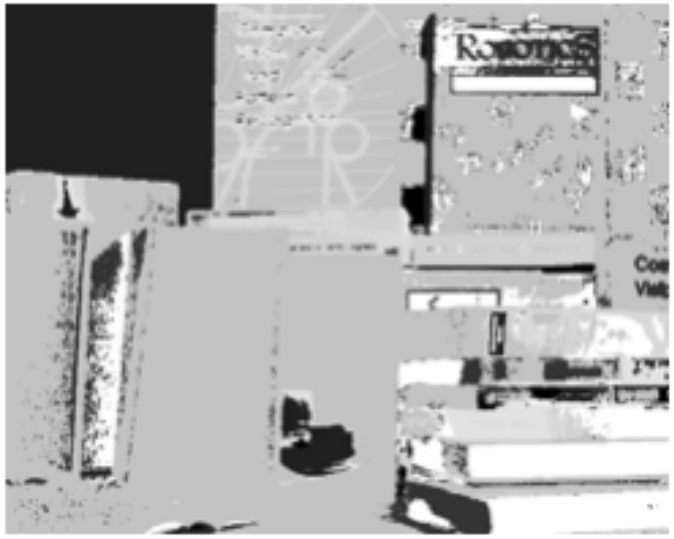


int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

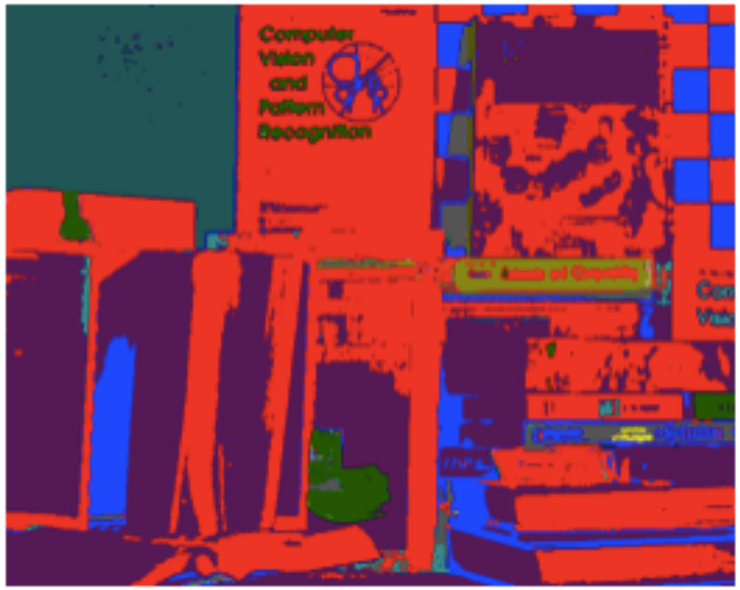


int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

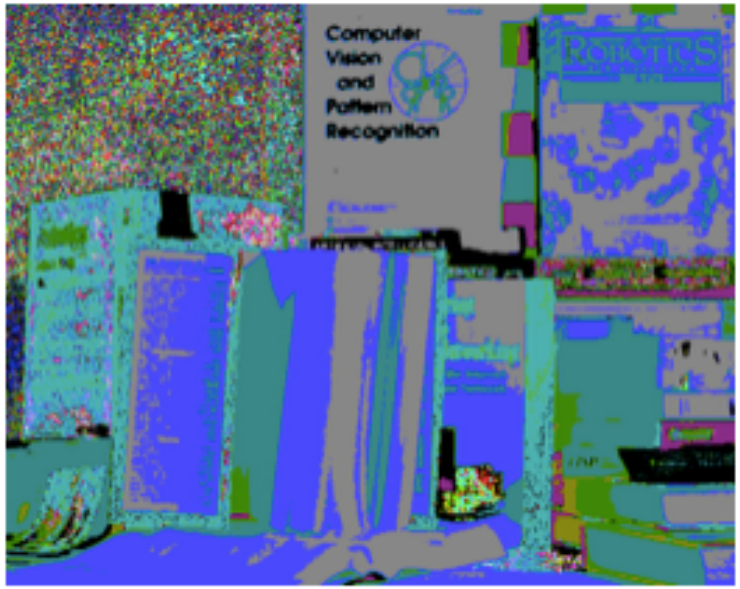


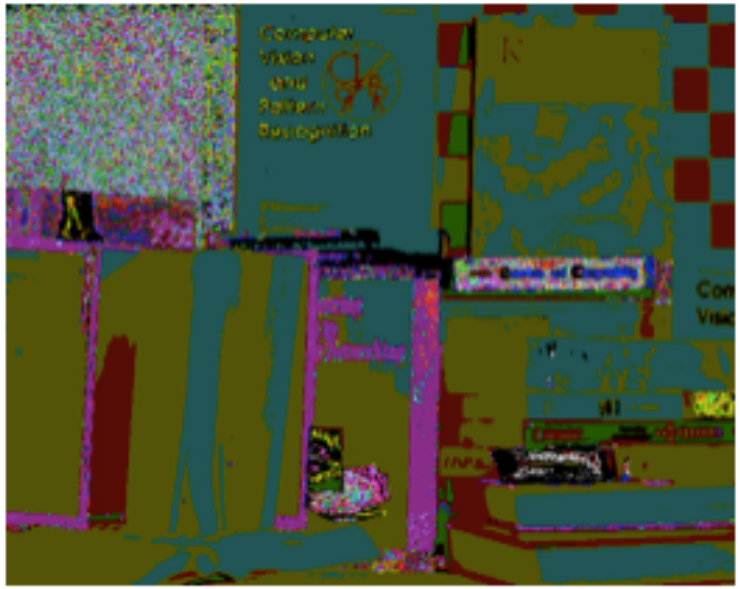


imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);

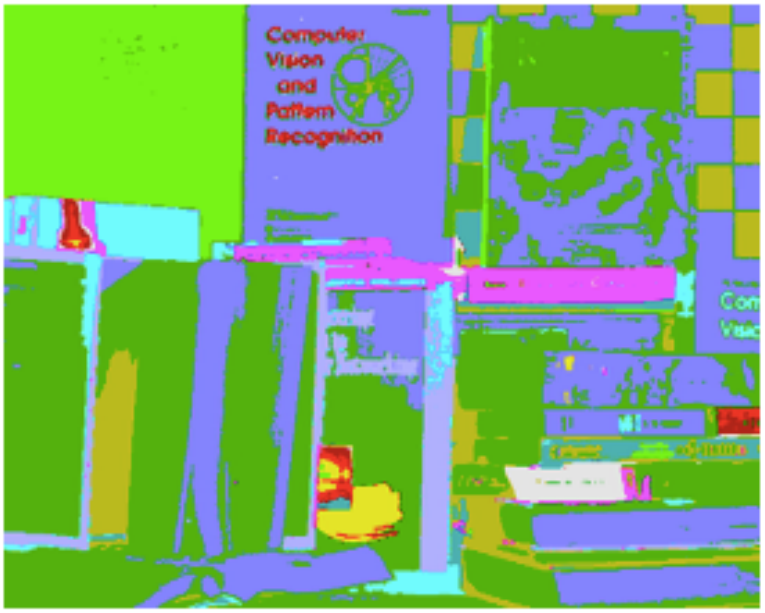


imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);

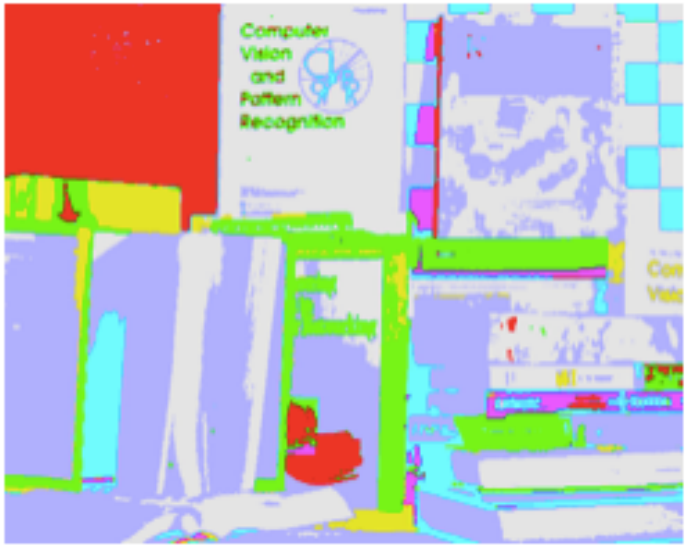




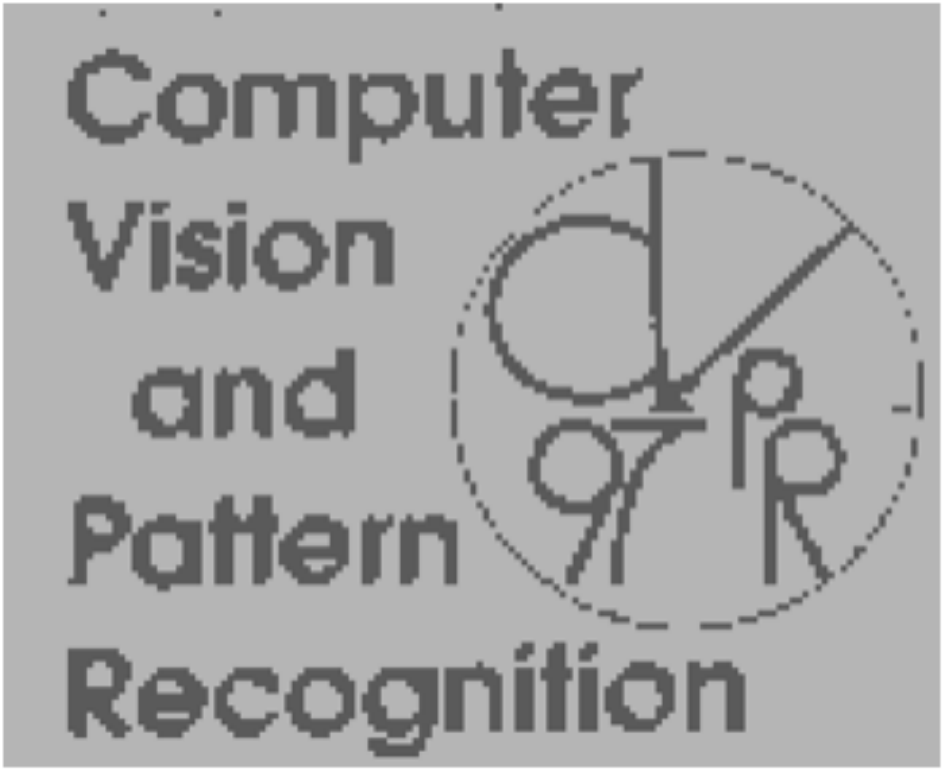
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);

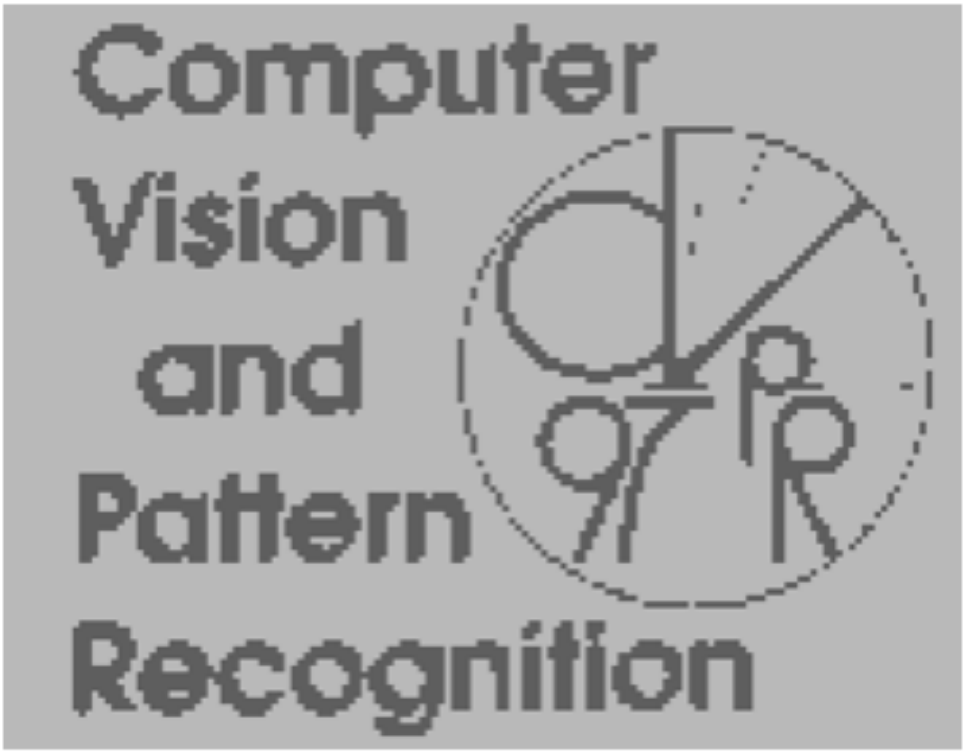


imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);

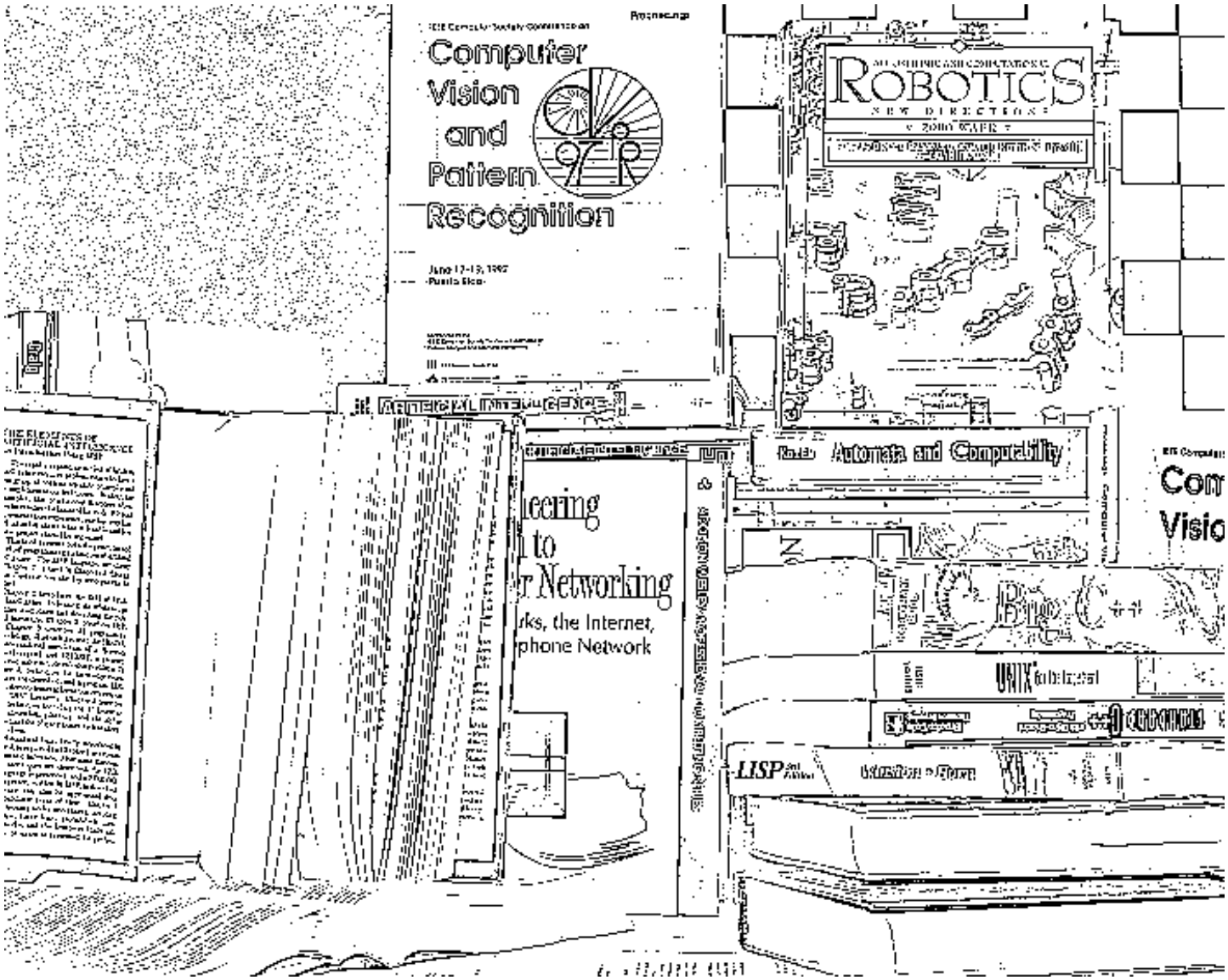


int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
zoom in

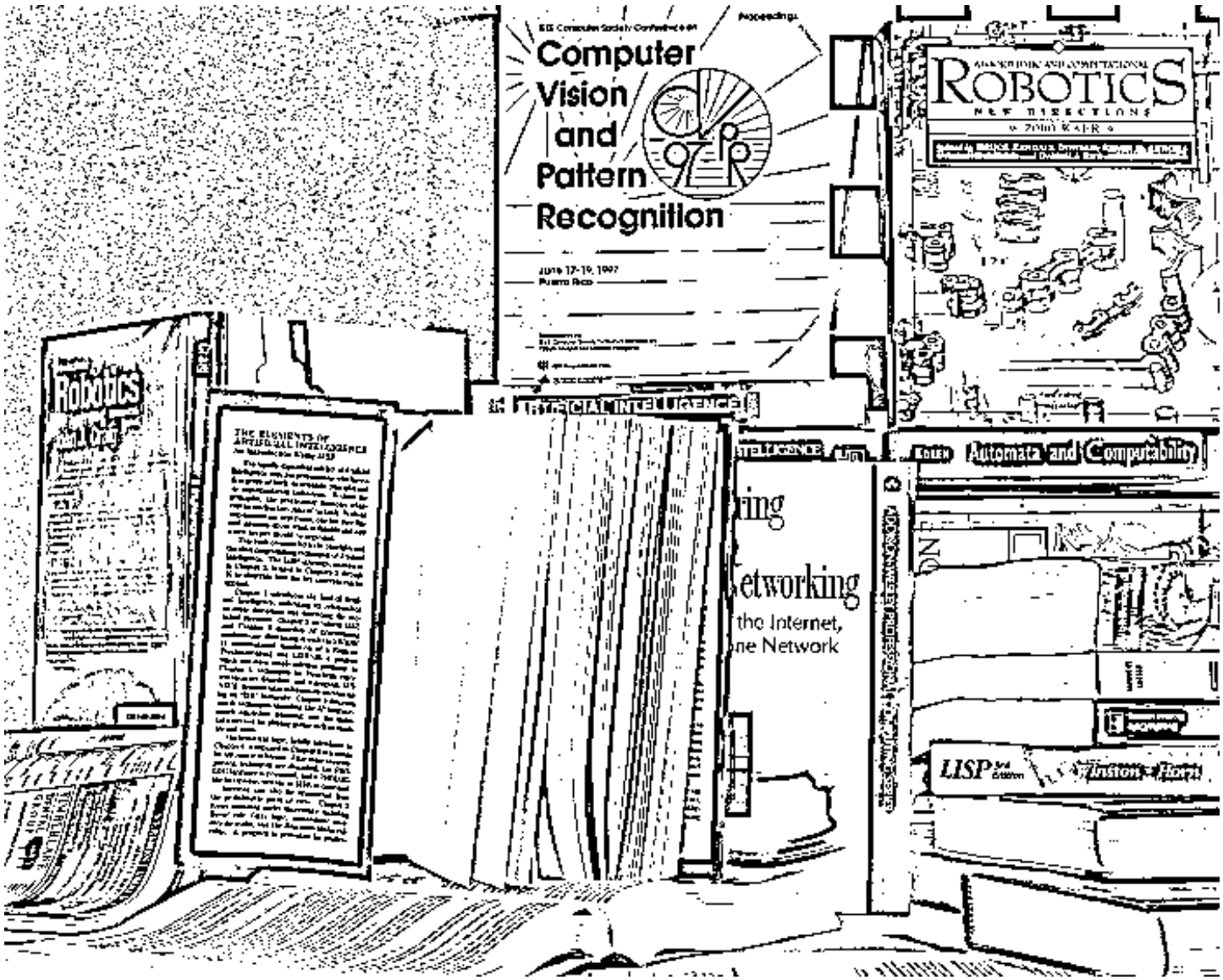




adaptive means, h=1

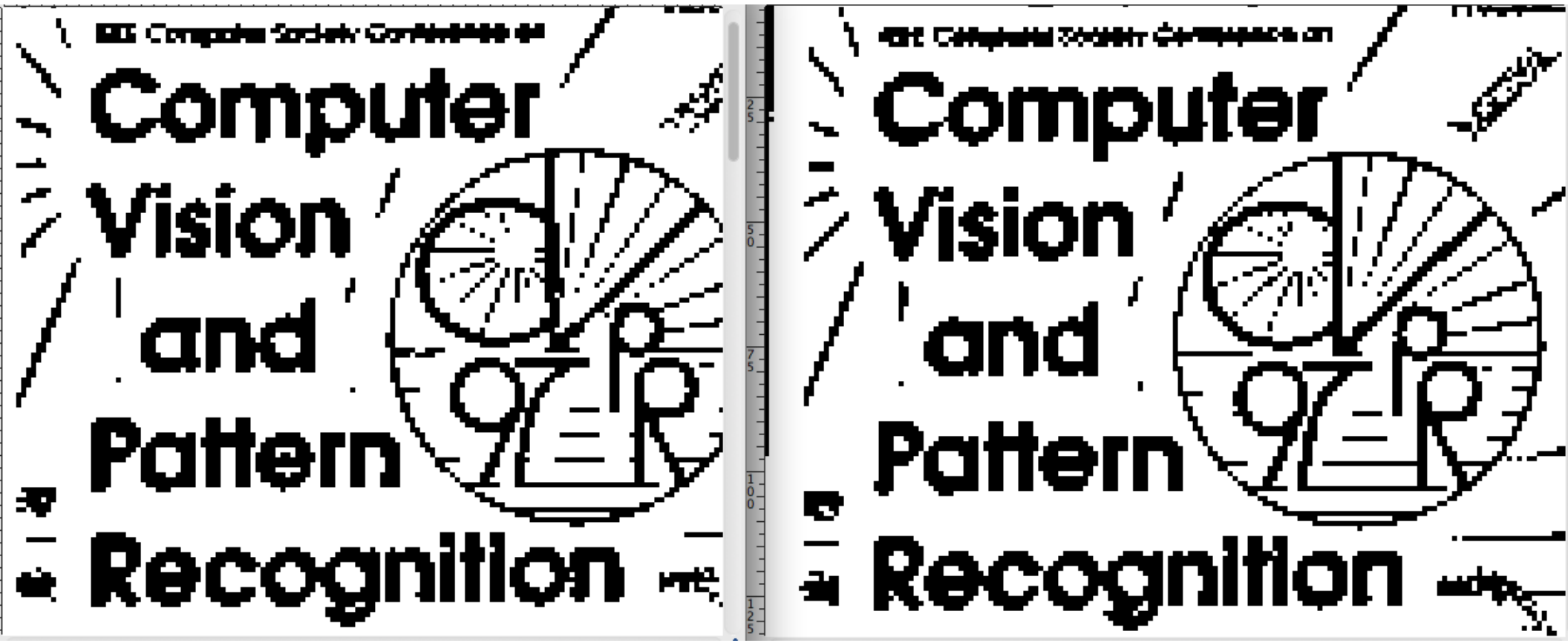


adaptive means, h=3

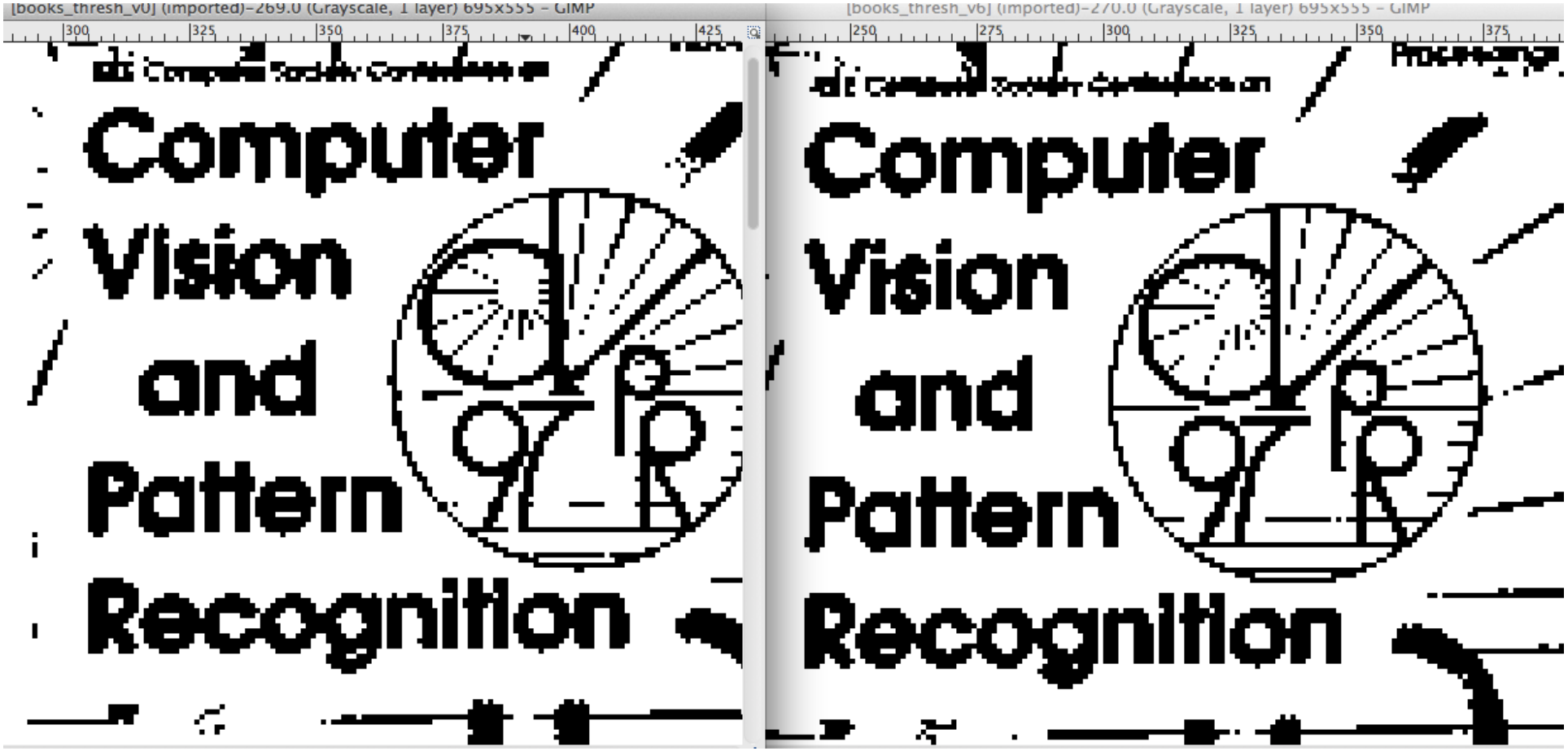


adaptive means, h=5

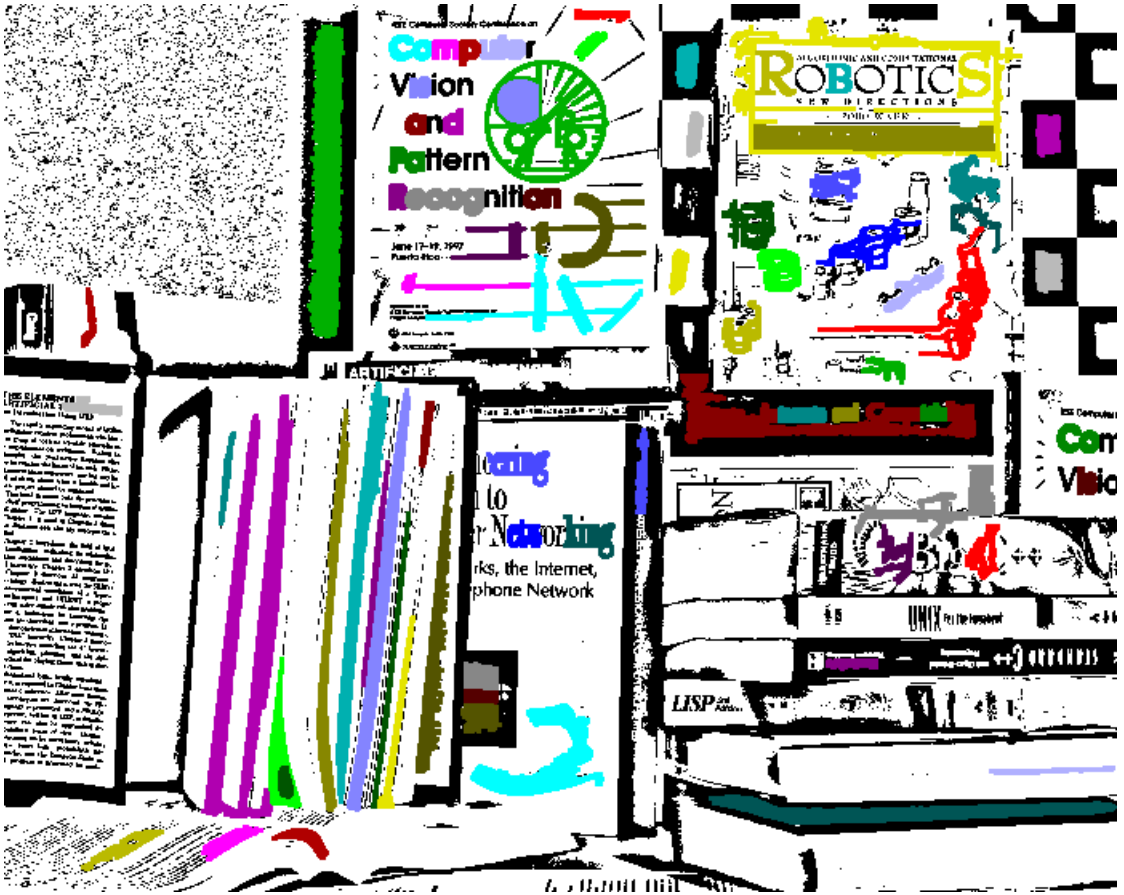
adaptive means, h=7



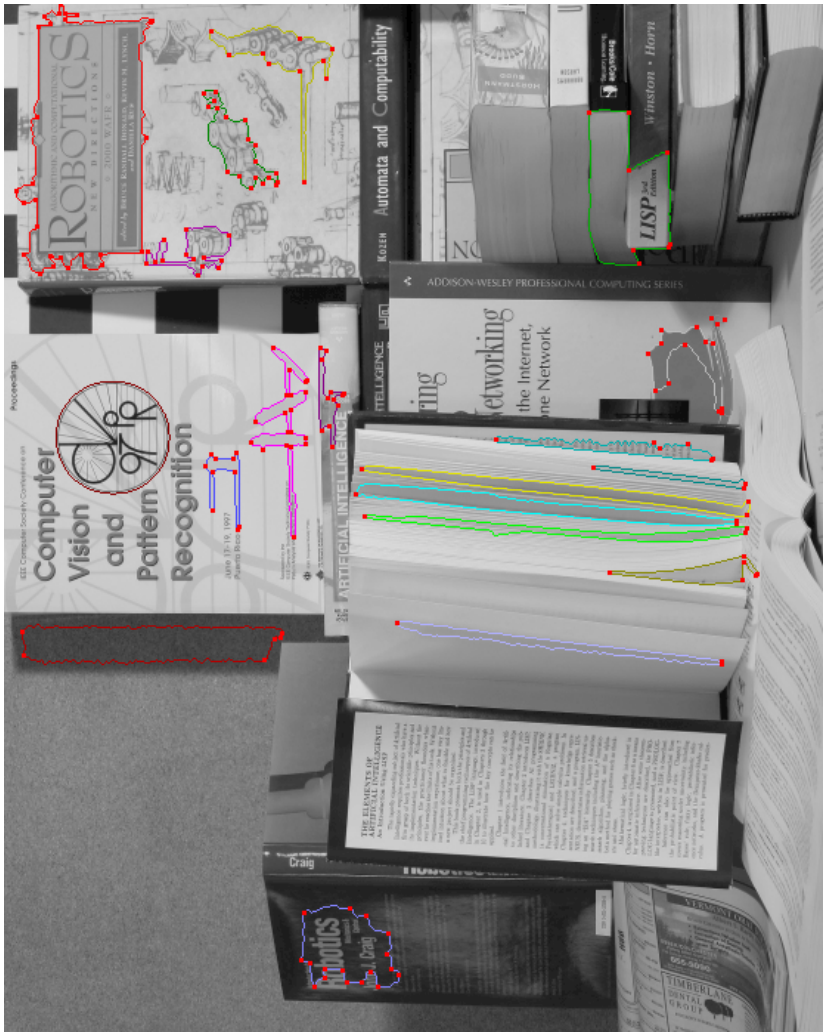
adaptive means, h=11 for adaptive to join close letters

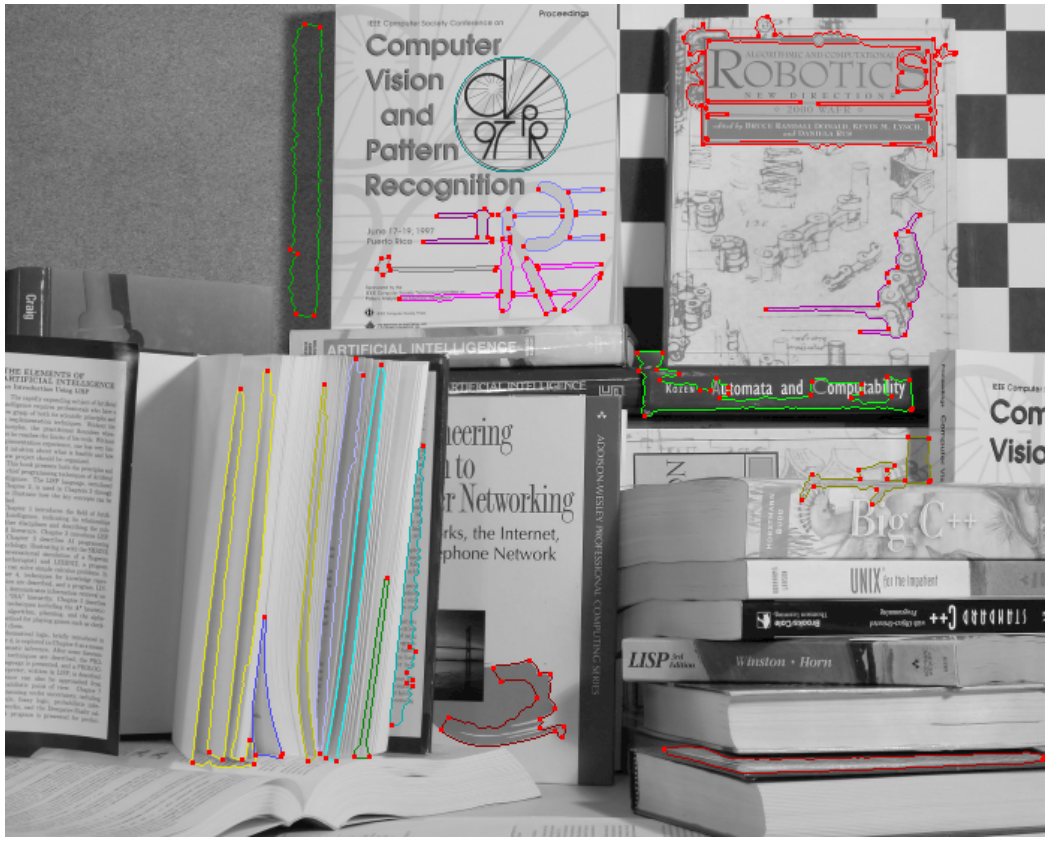


adaptive means, h=11 blobs (extr using scale calc code)



adaptive means, h=11 blob corners

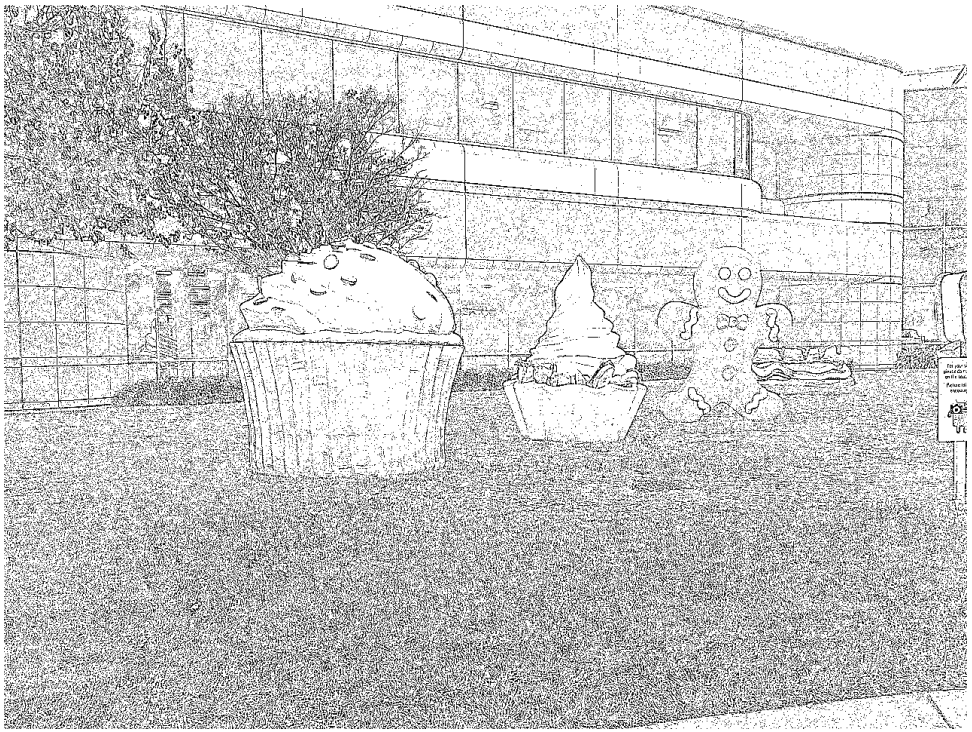




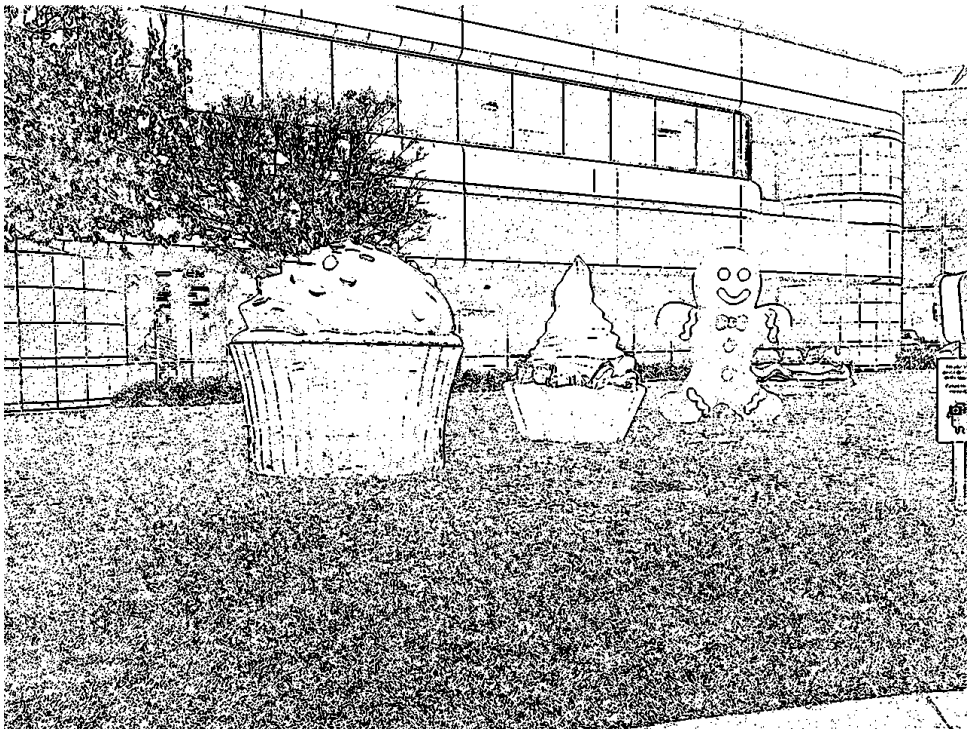
the letters are filtered out due to large nPerimeter compared to nInternal

adaptive means, h=3 was better for letters as blobs

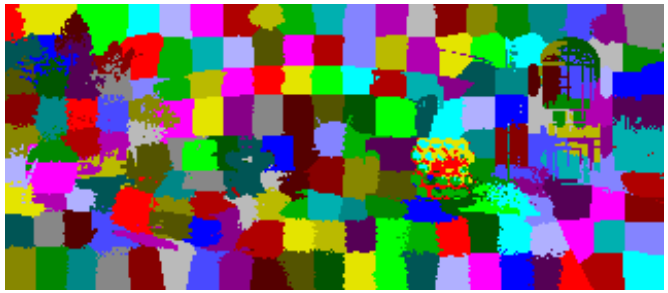
adaptive means, h=1

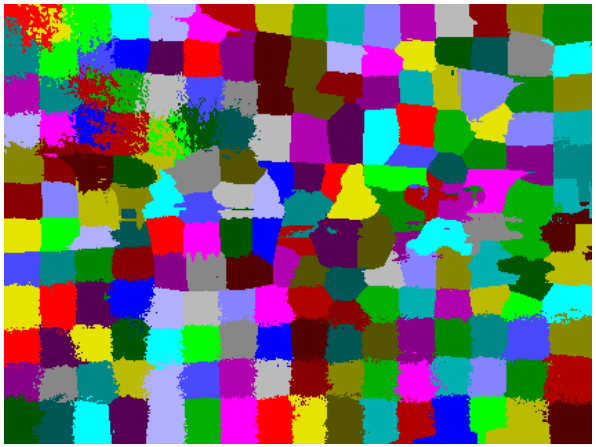
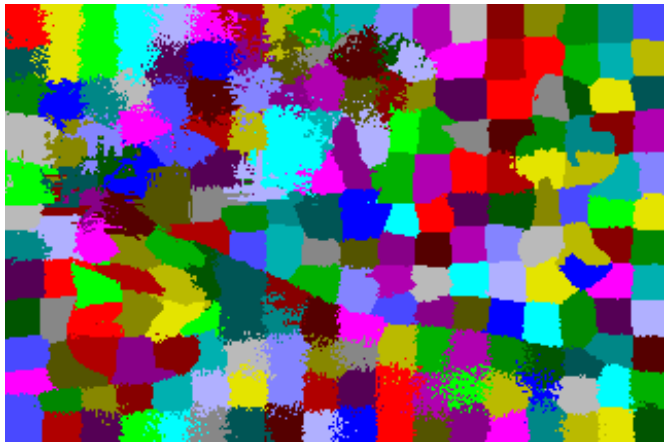


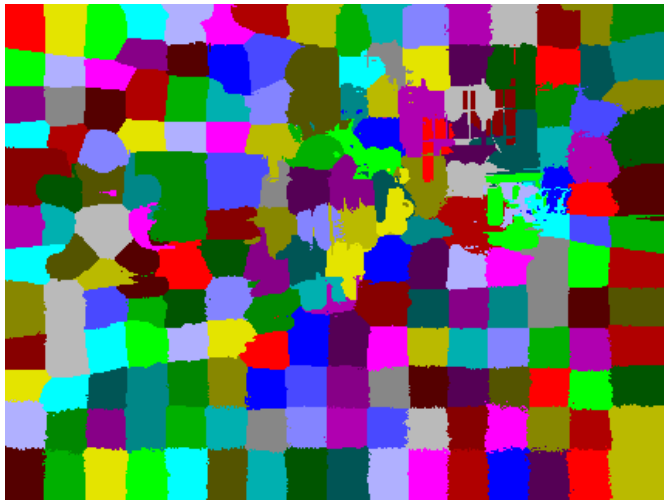
two median smooth with h=2



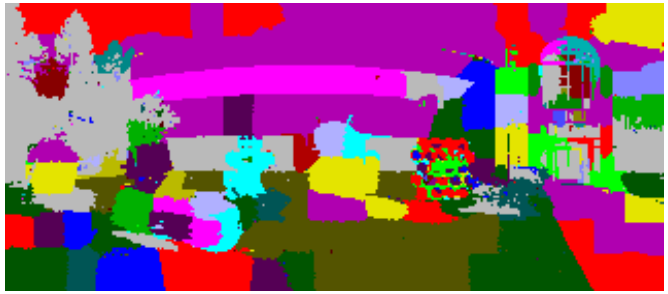
SLIC super pixels algorithm

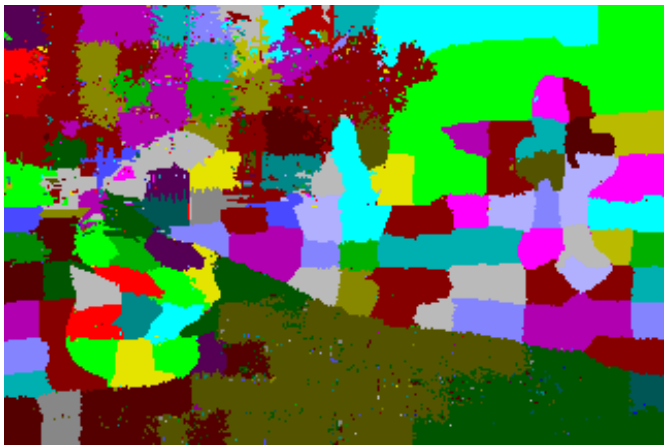


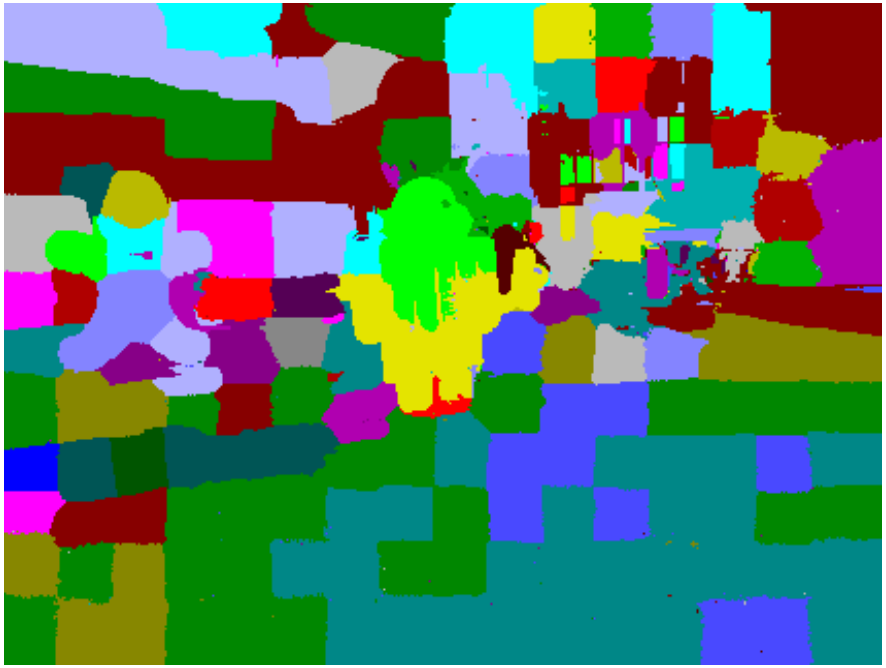



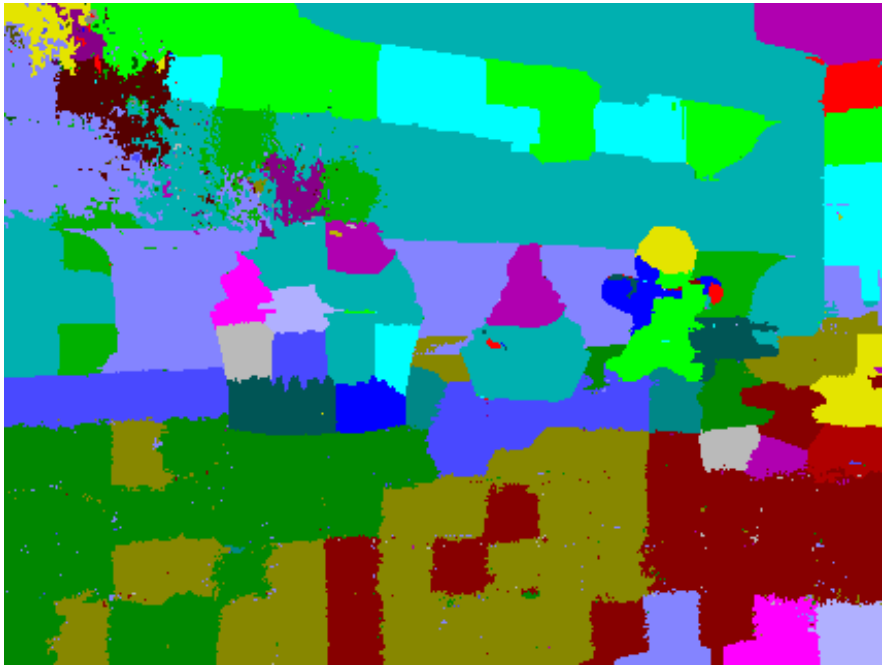




CIE XY Theta Clustering of the super pixels

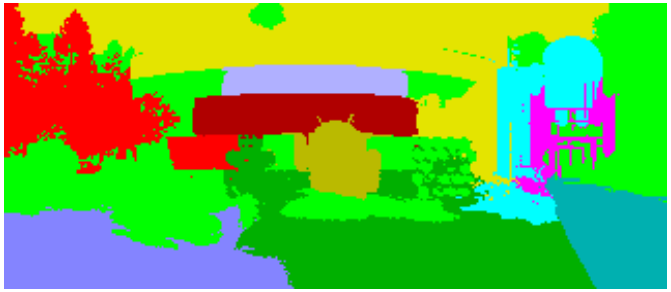


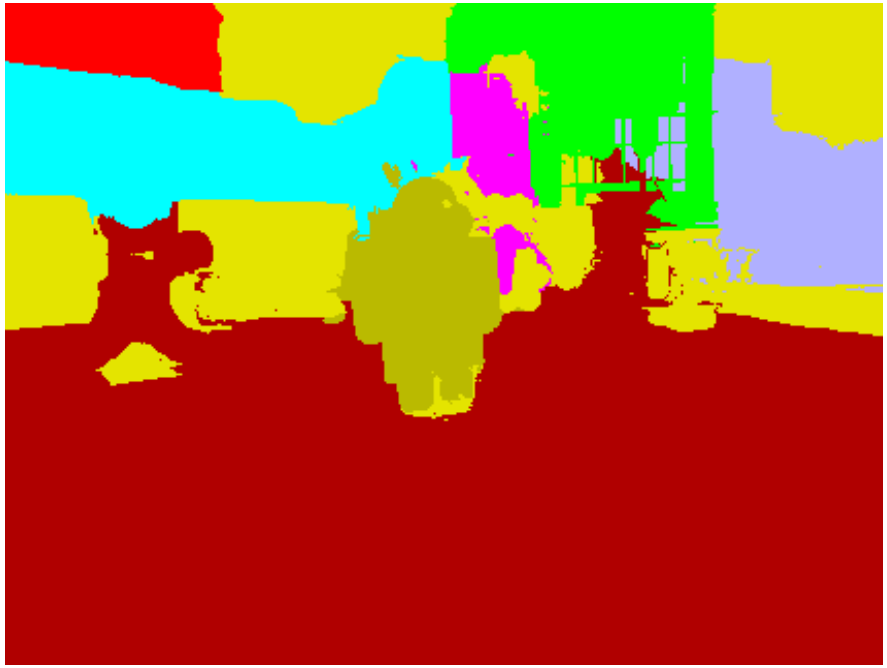


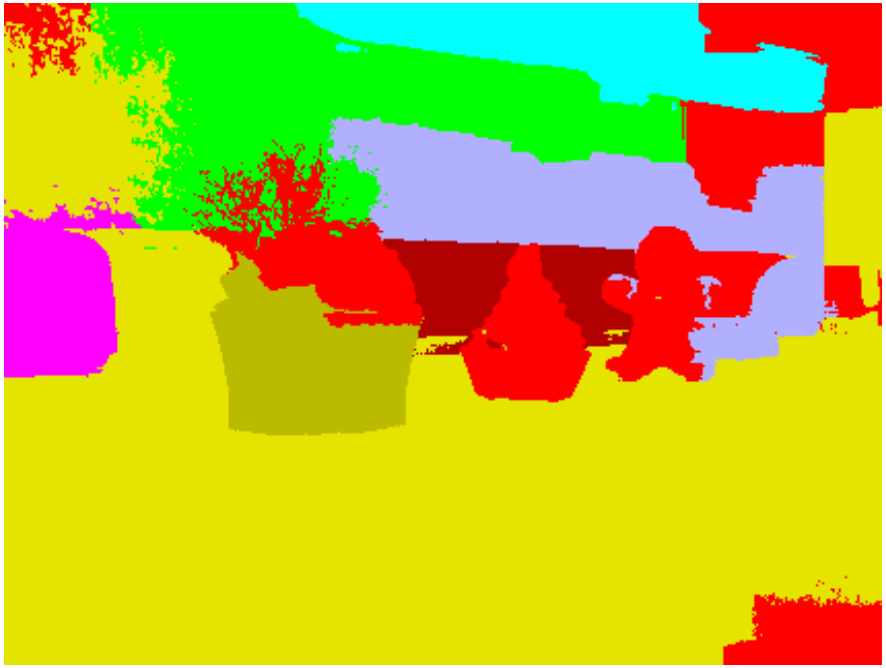




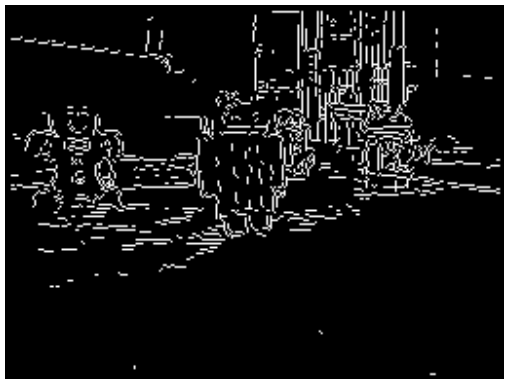
Normalized Cuts w/ HSV cs on the super pixels



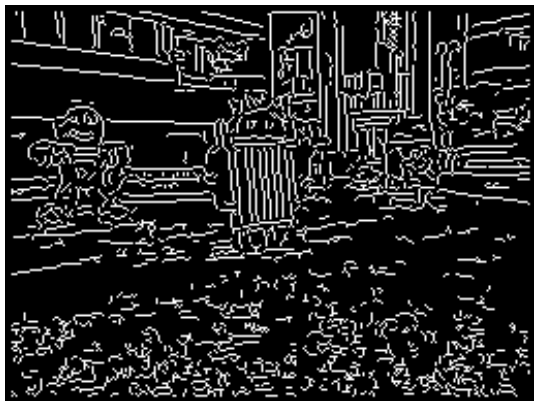


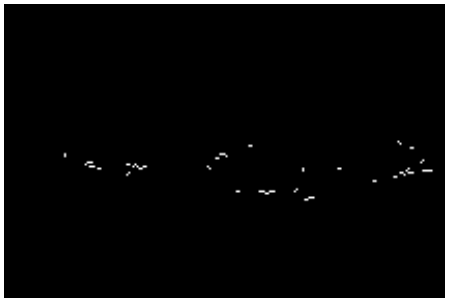


canny greyscale

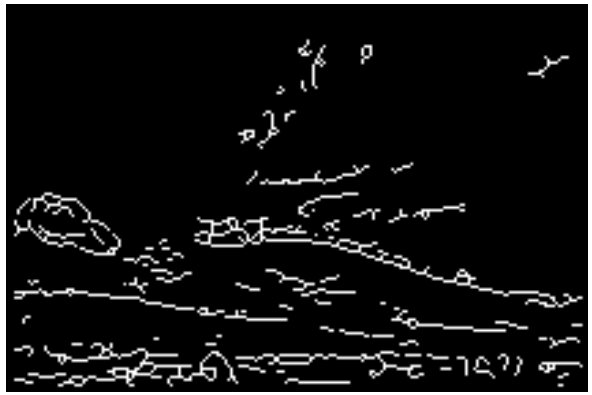


canny deltaE 2000

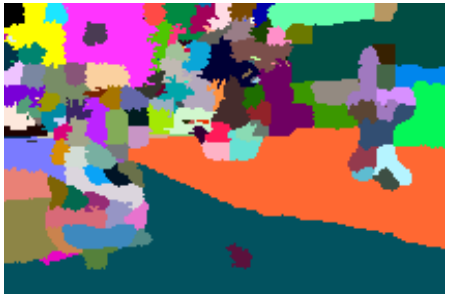


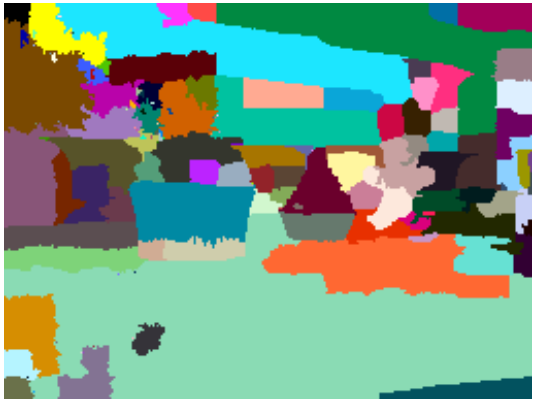


phase congruence



super pixels and moderate merging by HSV color

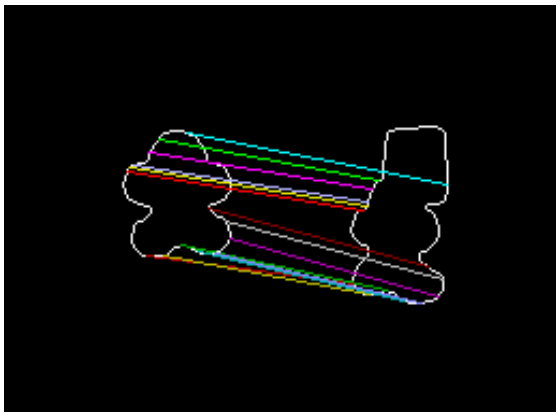



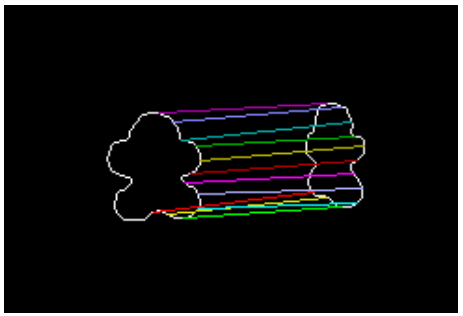


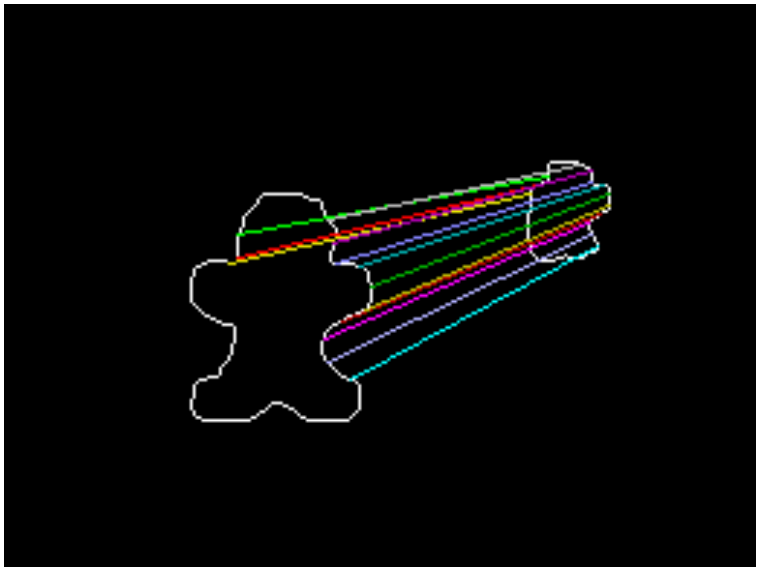


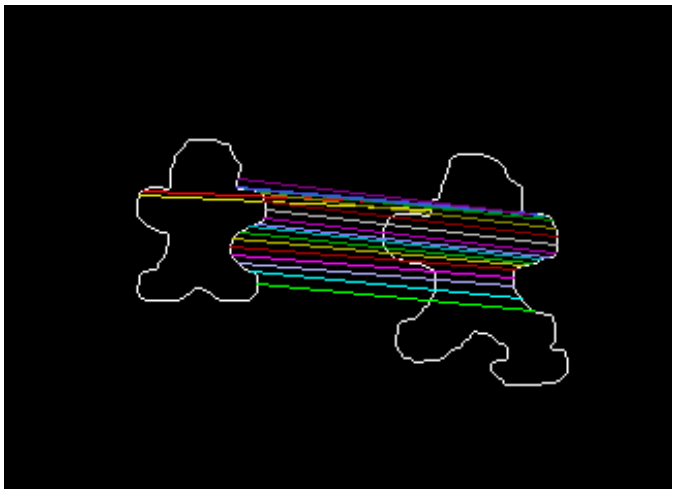
Partial Shape Matching of the gingerbread man

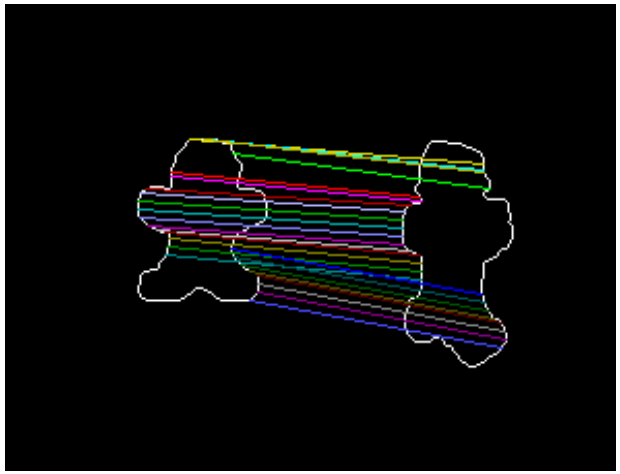
extract ordered borders of shapes, blur them by one or two sigma, then ordered bipartite matching algorithm using difference of chords as a descriptor.

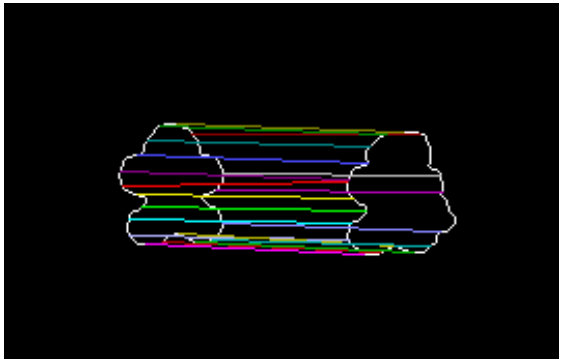












Shape Finding in over-segmented images

using the partial shape matcher and a search pattern (which will be improved), to make progress towards finding the gingerbread man.

Finding the gingerbread man with just descriptors within the test images which have different lighting, poses and background did not work well *demonstrated in other notes) - could see that matching groups of points was necessary. Also found that if descriptors are used, they must exclude the background.

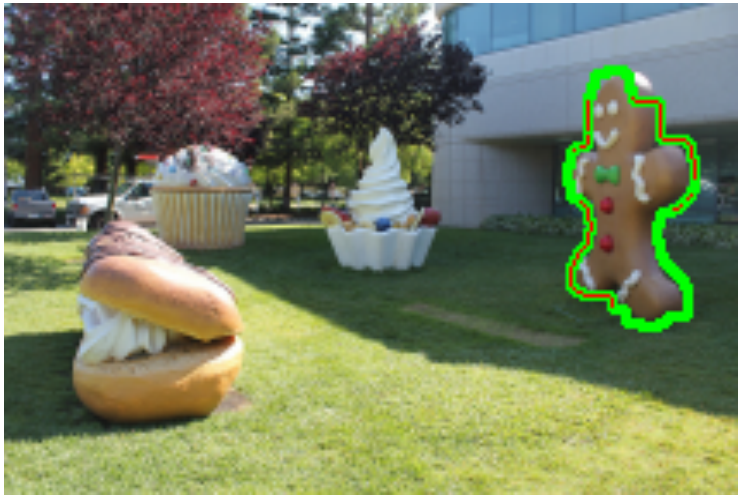
Shown below and on next page are a test image which has been processed to make super-pixels, then a merge of super-pixels by HSV color, then a CIE DeltaE 2000 color filter (difference from template gingerbread man) applied to remove some of the segmented cells.

The search results are preliminary and still a work in progress, but can see that if more information about color or texture is used, the true match will be found as top match. NOTE that a histogram of which segmented cells are present in the top k answers might be a way to find the true match without additional descriptors. see the *matched*png images in the output (next).

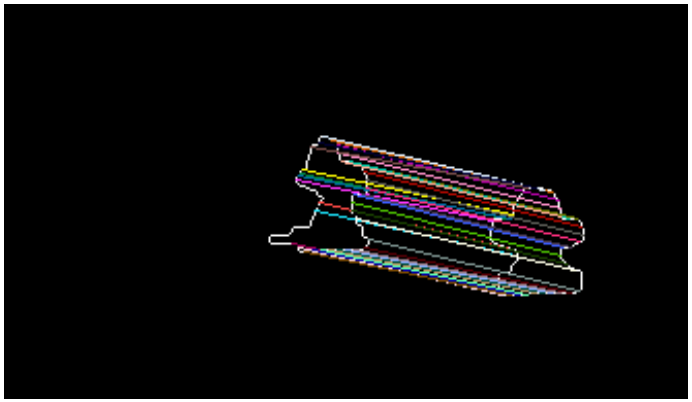
top matches to template shape within a test image

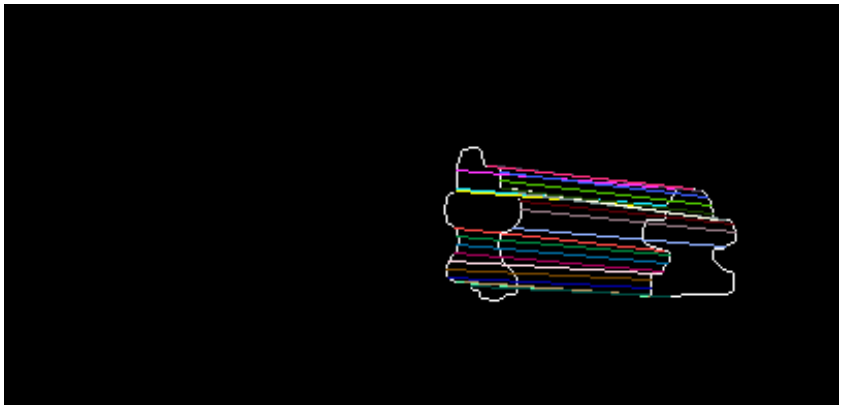
best match **and** **true match**

the min cost match to the template shape



the true match has higher cost. (due to scale, and pose)





Shape Finding in over-segmented images (cont)

Can see from rough results that searching over different template scales is necessary and additional information about color or texture might be necessary.

trying by just shape first to explore the ability to match just silhouettes. can see that the best match would not be found as a silhouette though, but should help in exploring reasons for false matches…