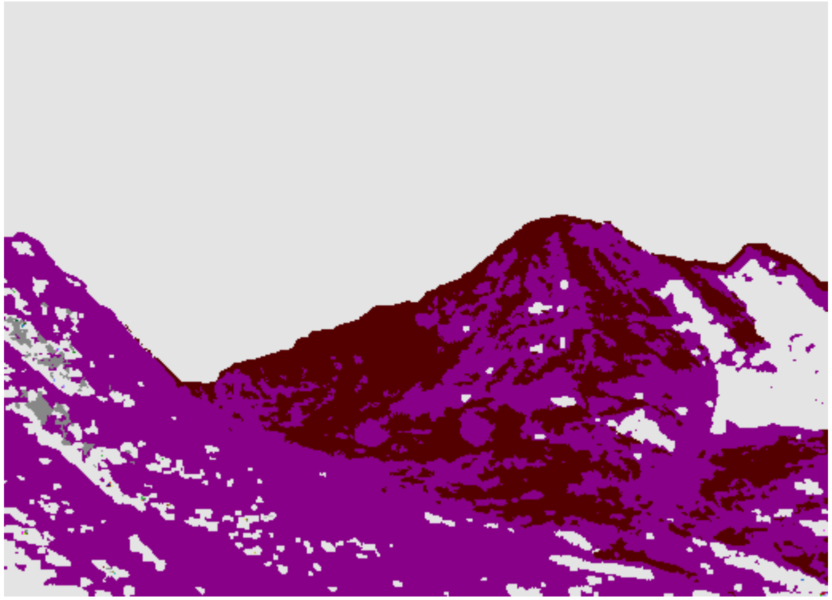
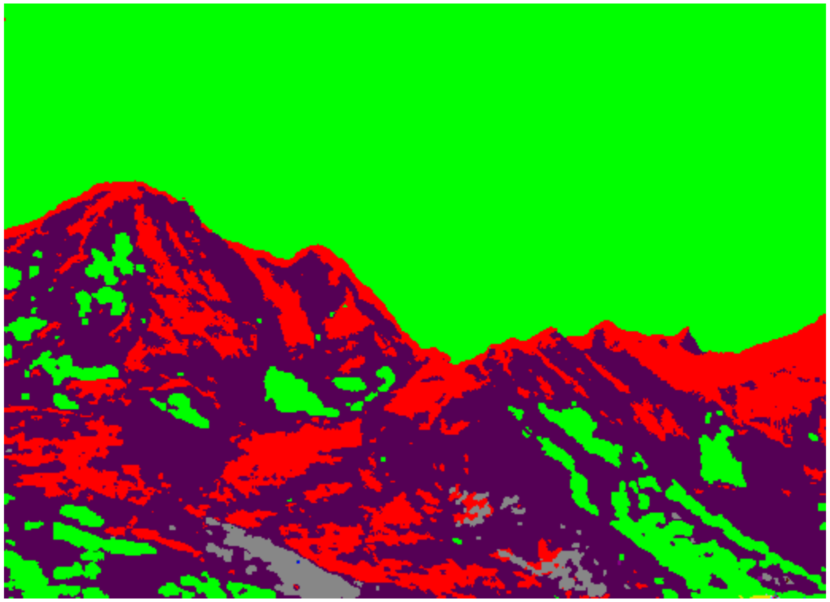
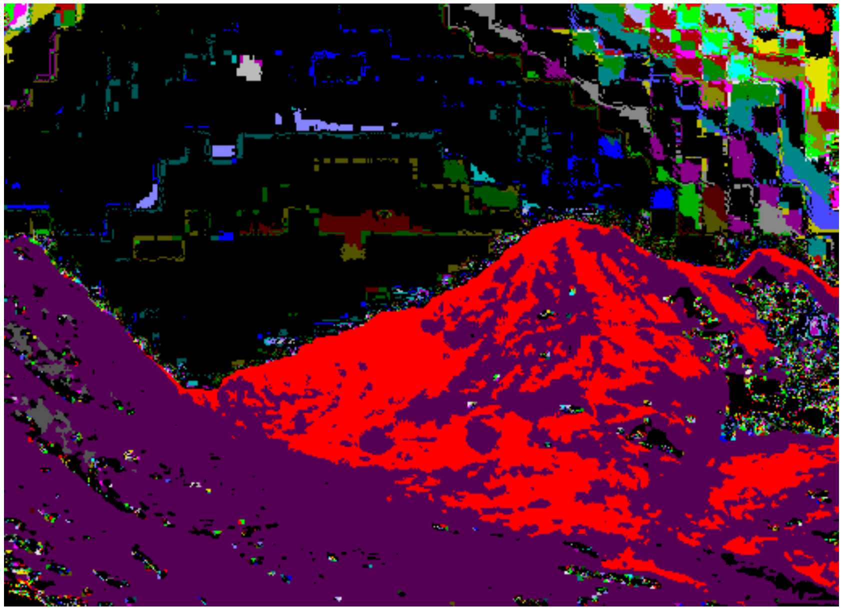
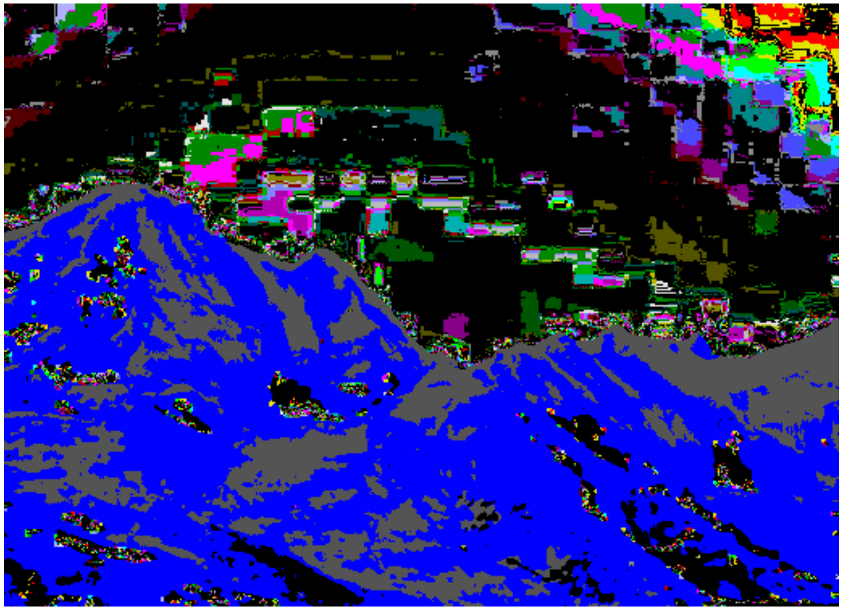
```
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);
```



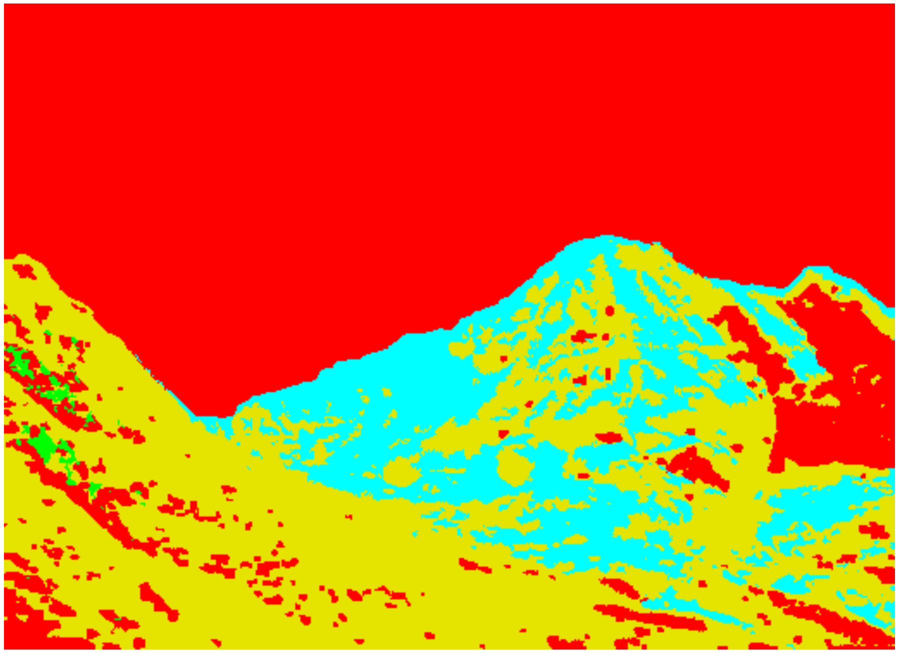


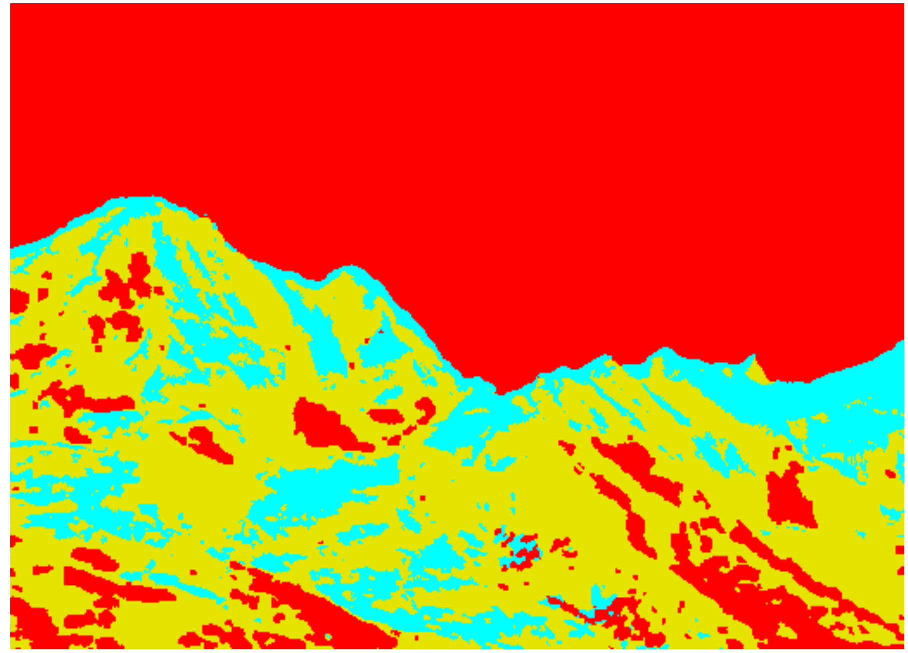
```
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);
```



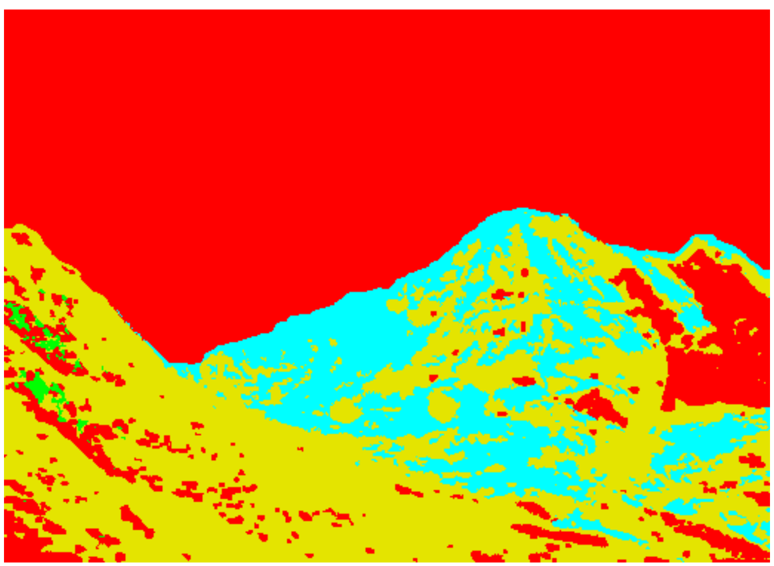


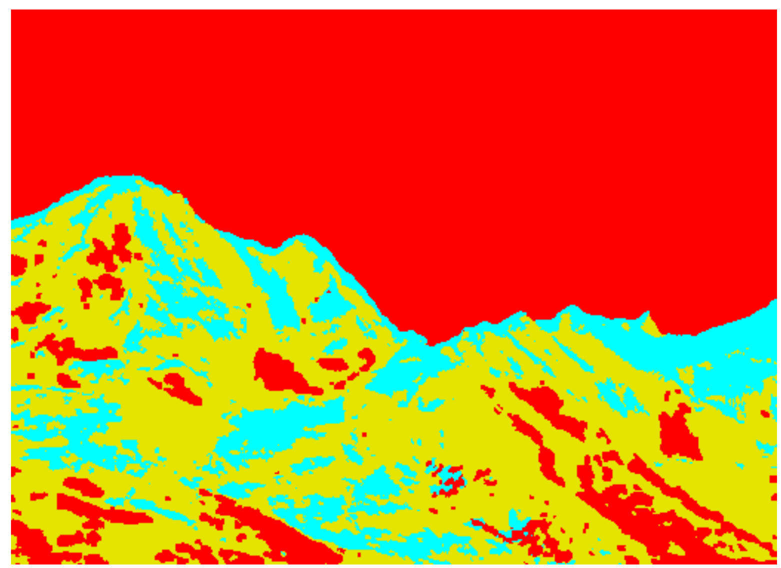
```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);
```





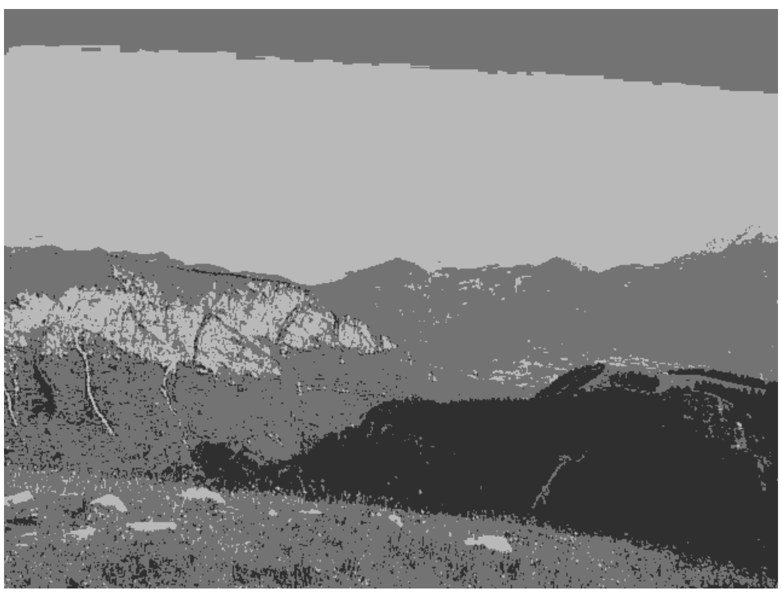
```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);
```





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

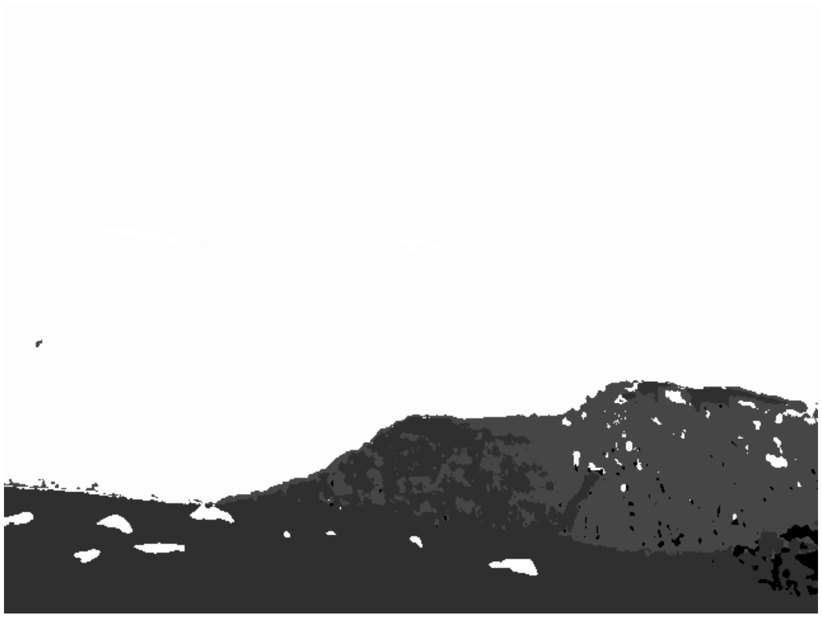
```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```



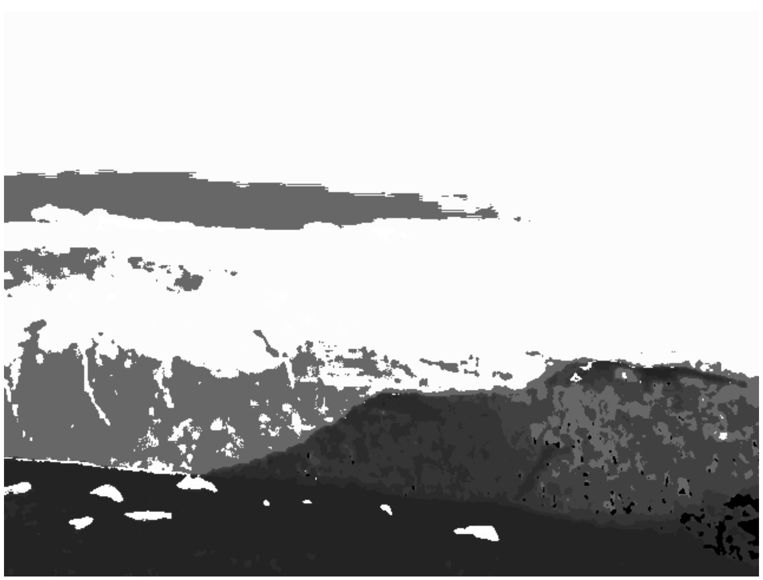
```
int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

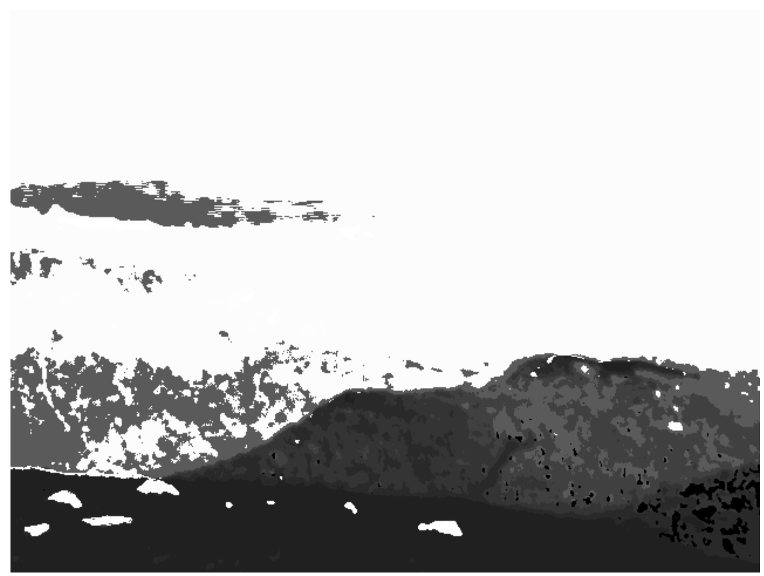
```
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);
```

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



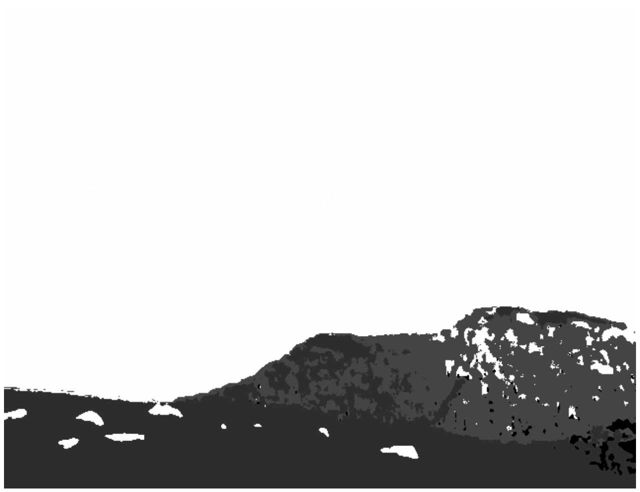
```
int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);
```

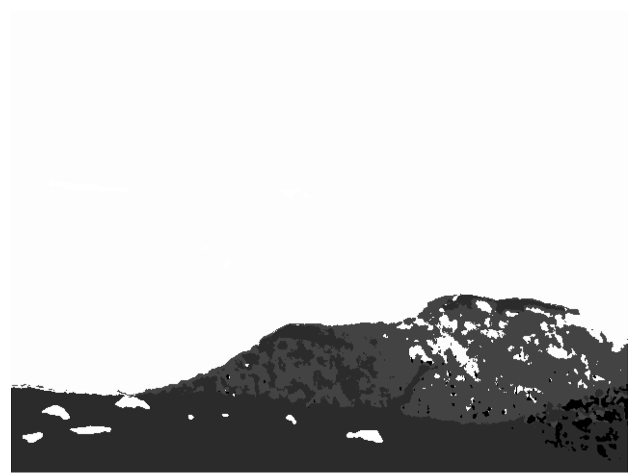




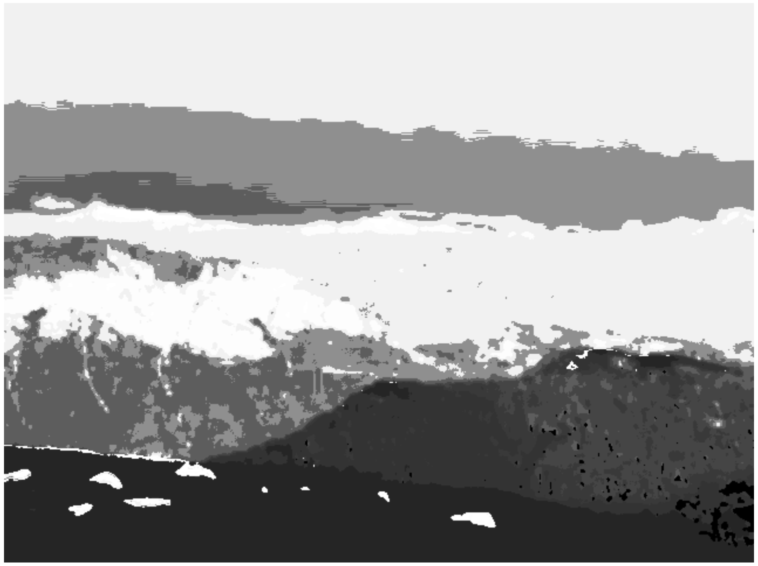
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

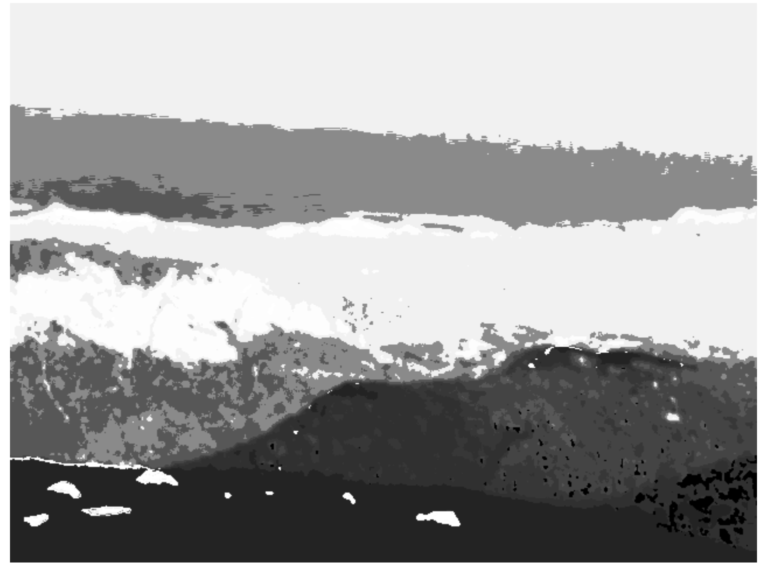
int kBands =3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);





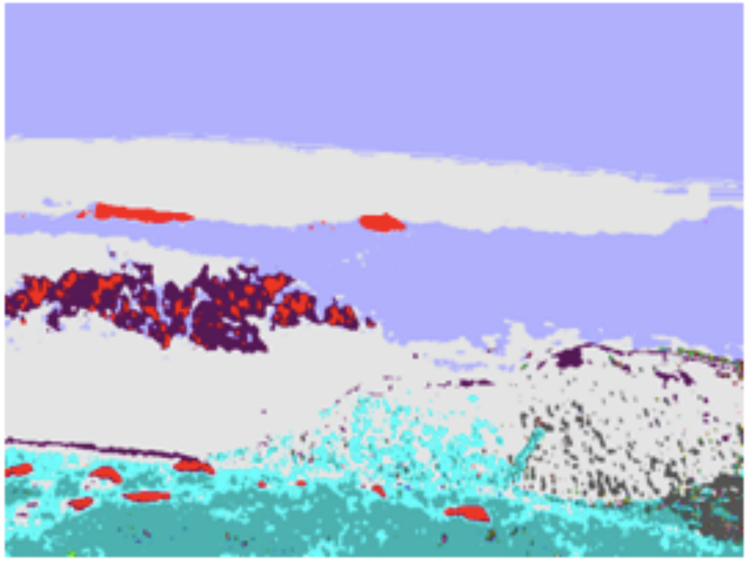
int kBands =8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq

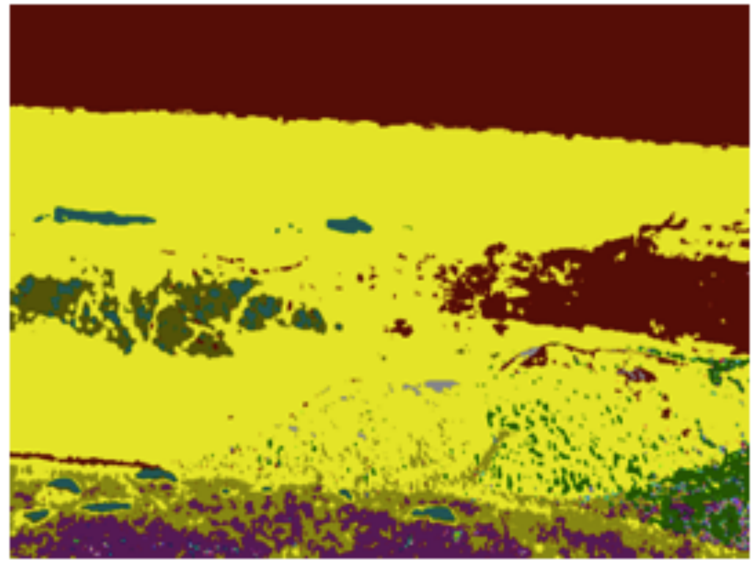




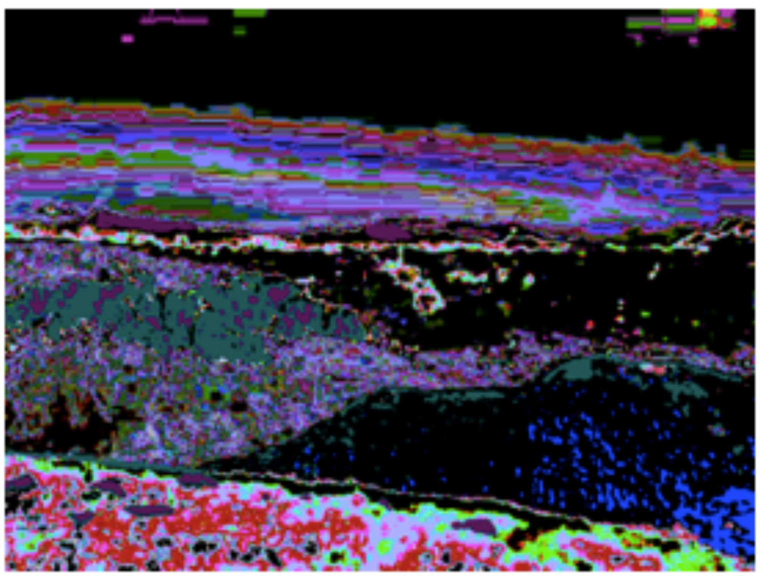
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistogram(gsImg1, kBands);

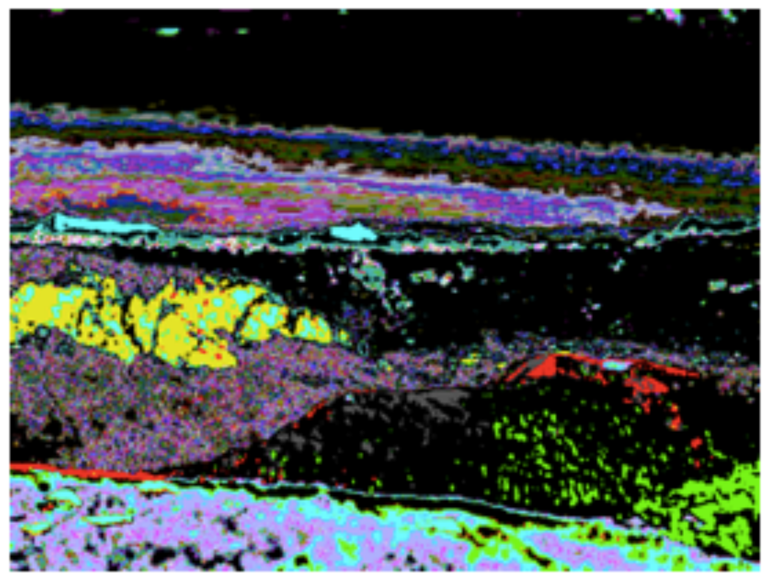
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);



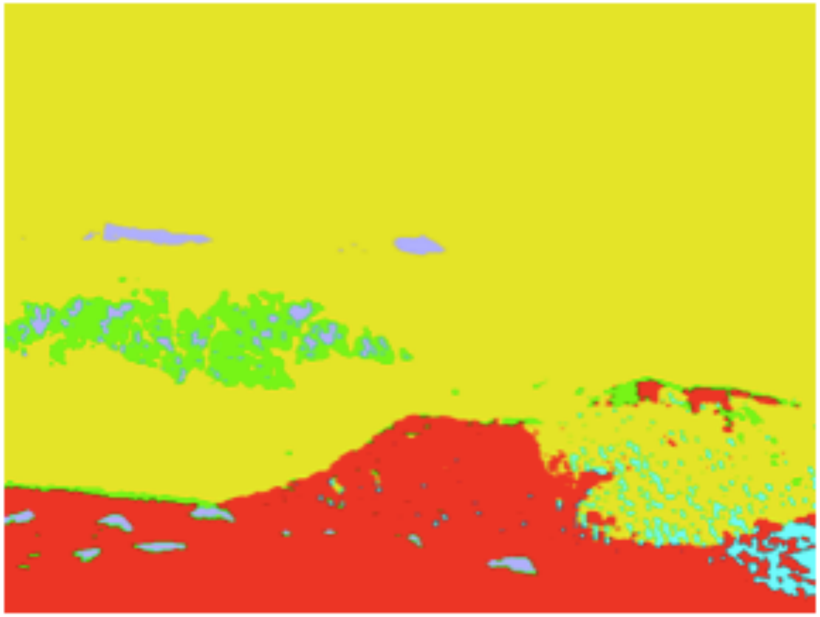


imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);

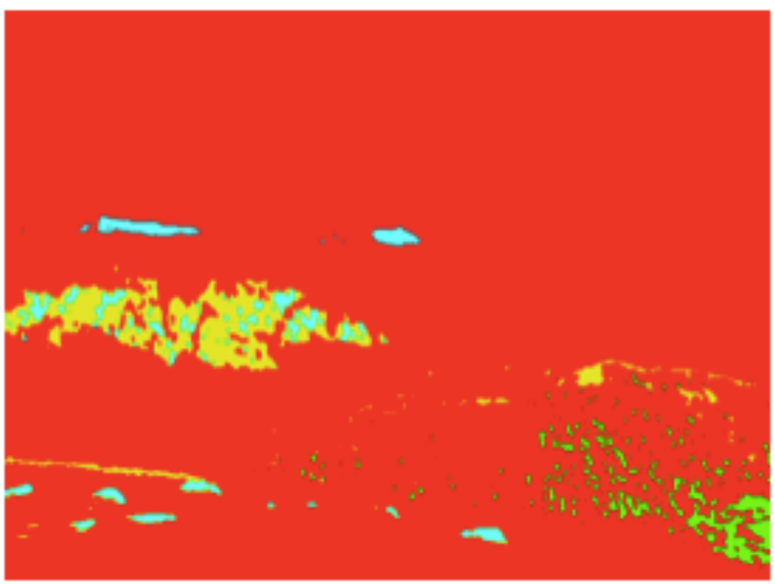


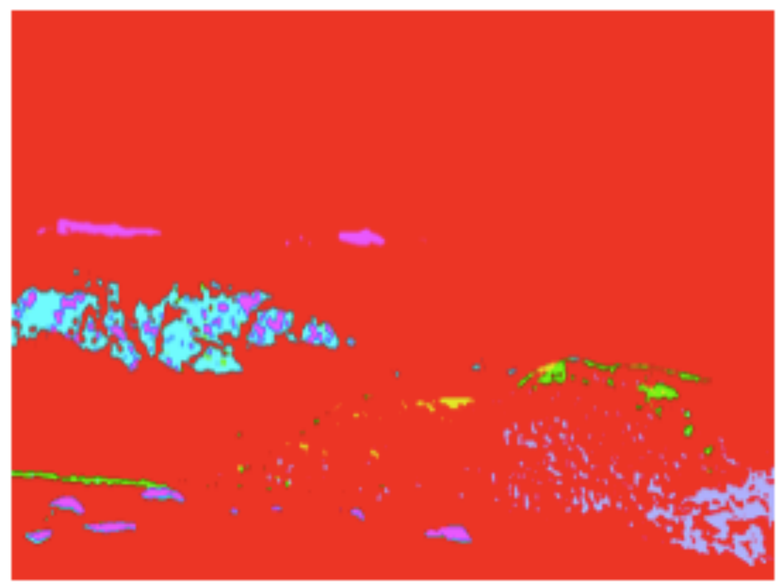


imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);



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





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

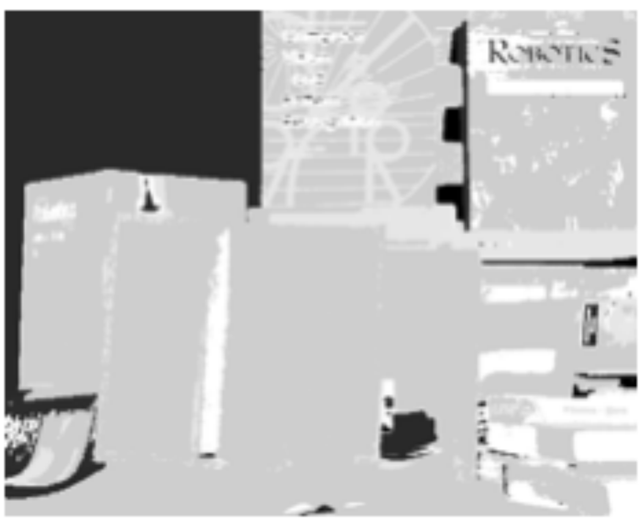

```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

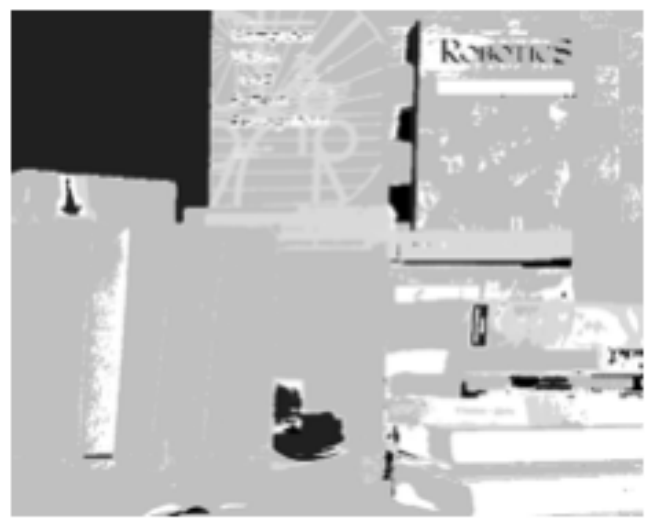
```
int kBands = 8; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```


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

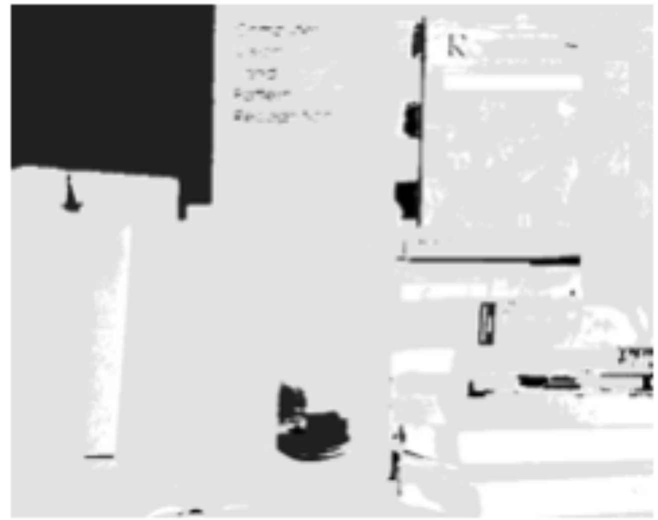
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

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



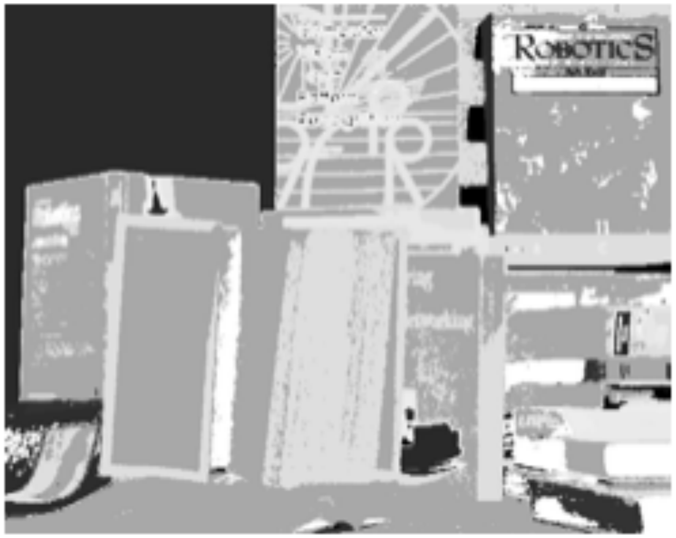


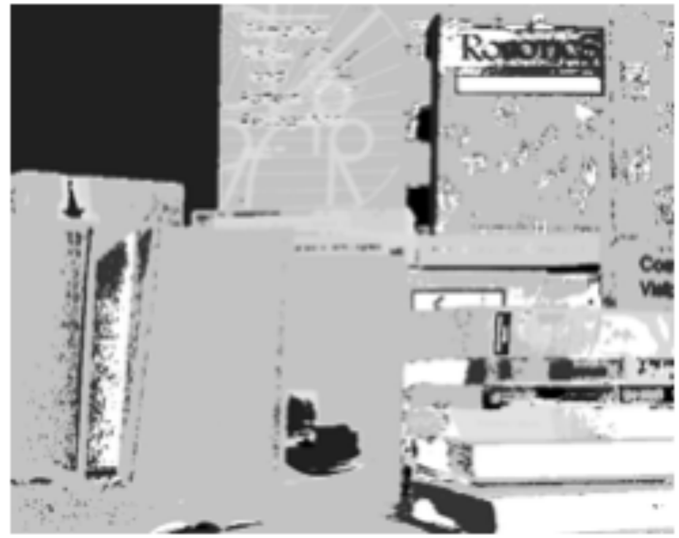
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);



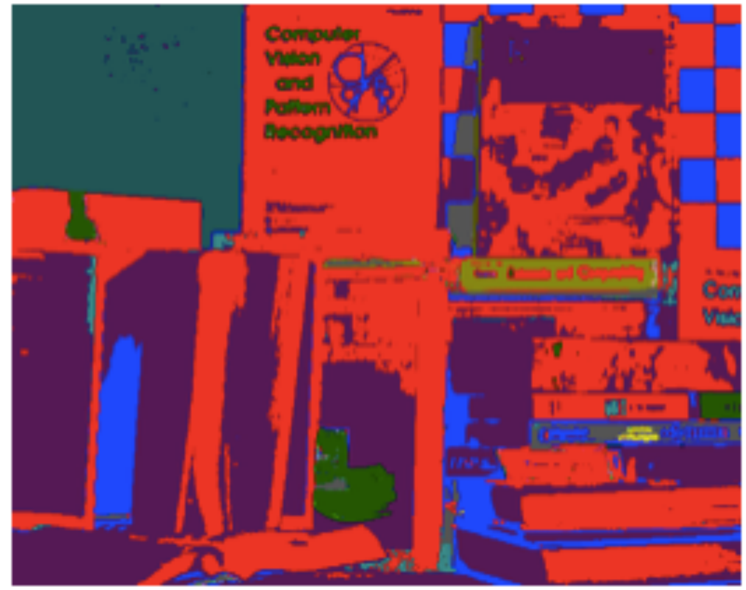
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

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

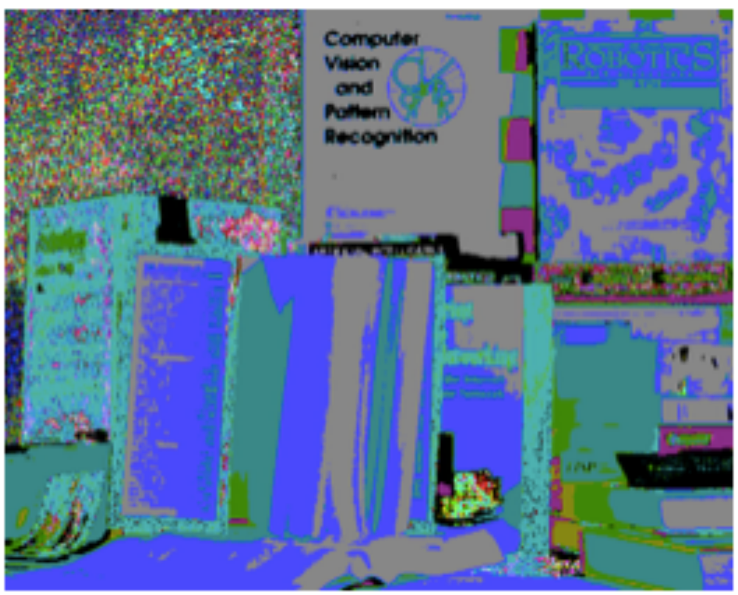


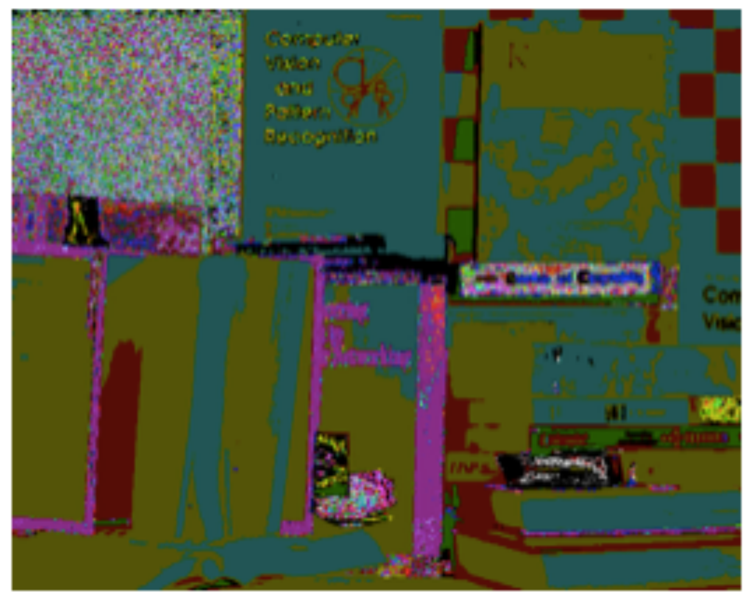


```
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);
```

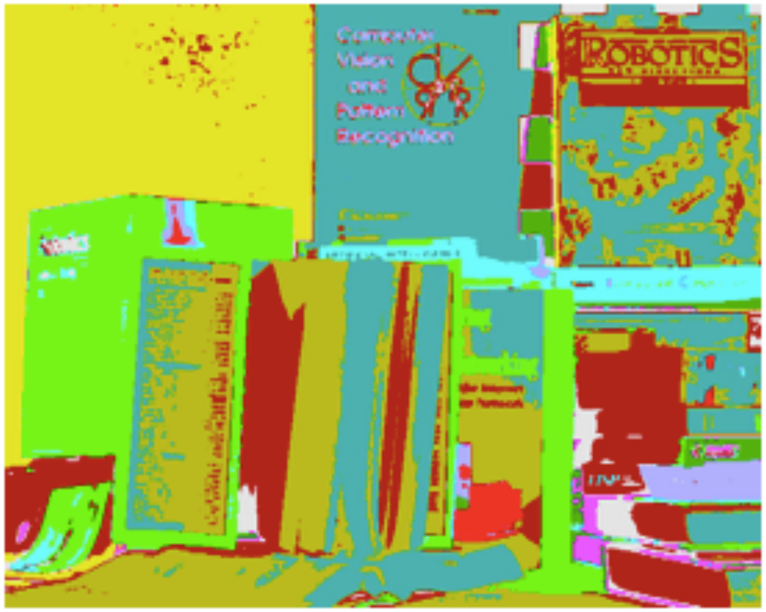


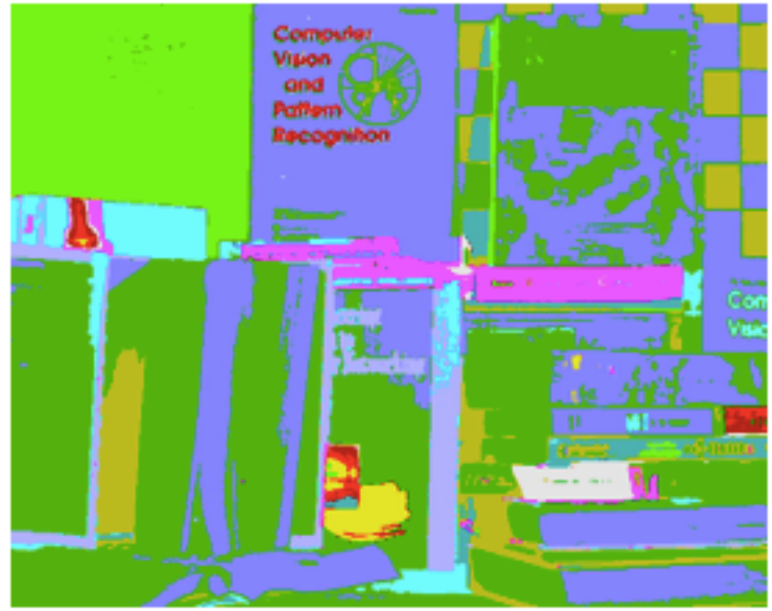

```
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);
```



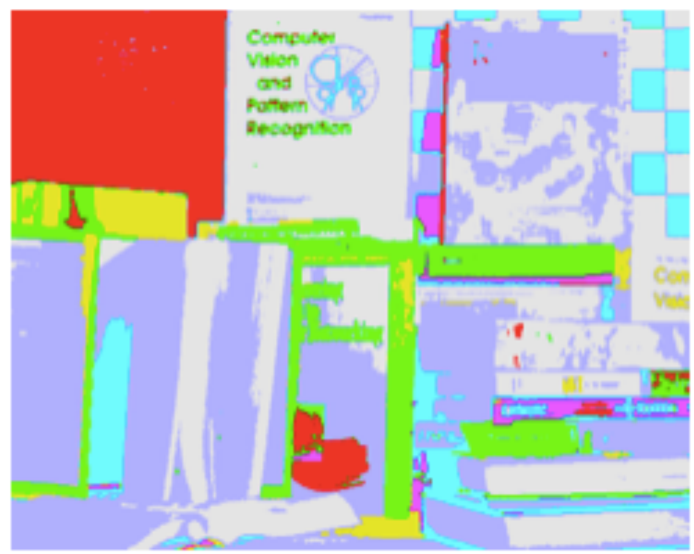


imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);

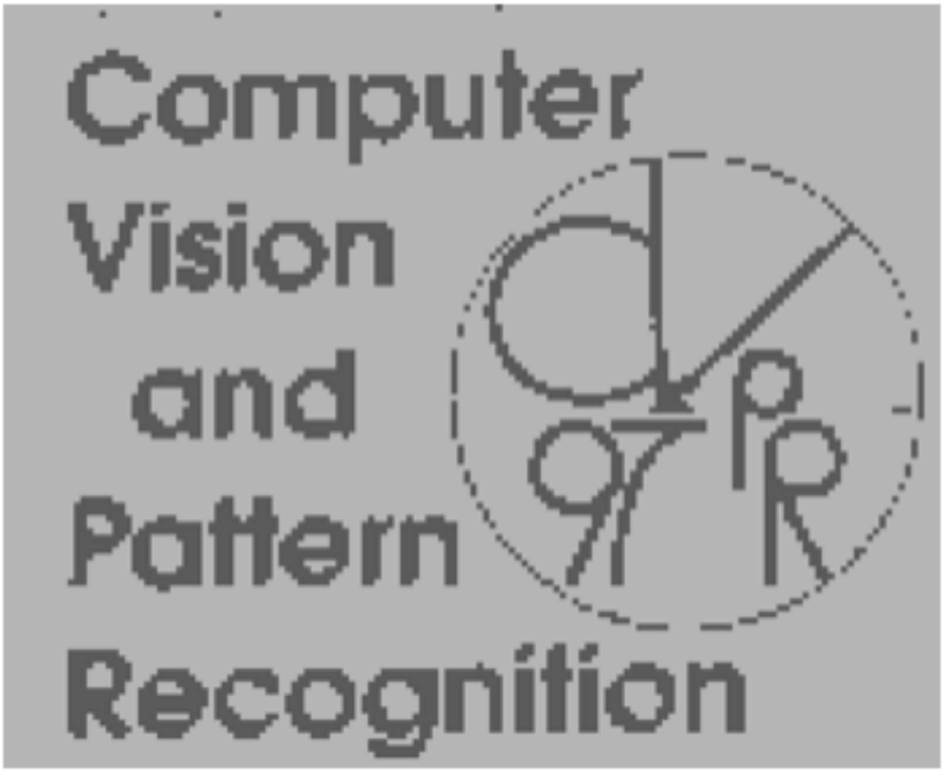


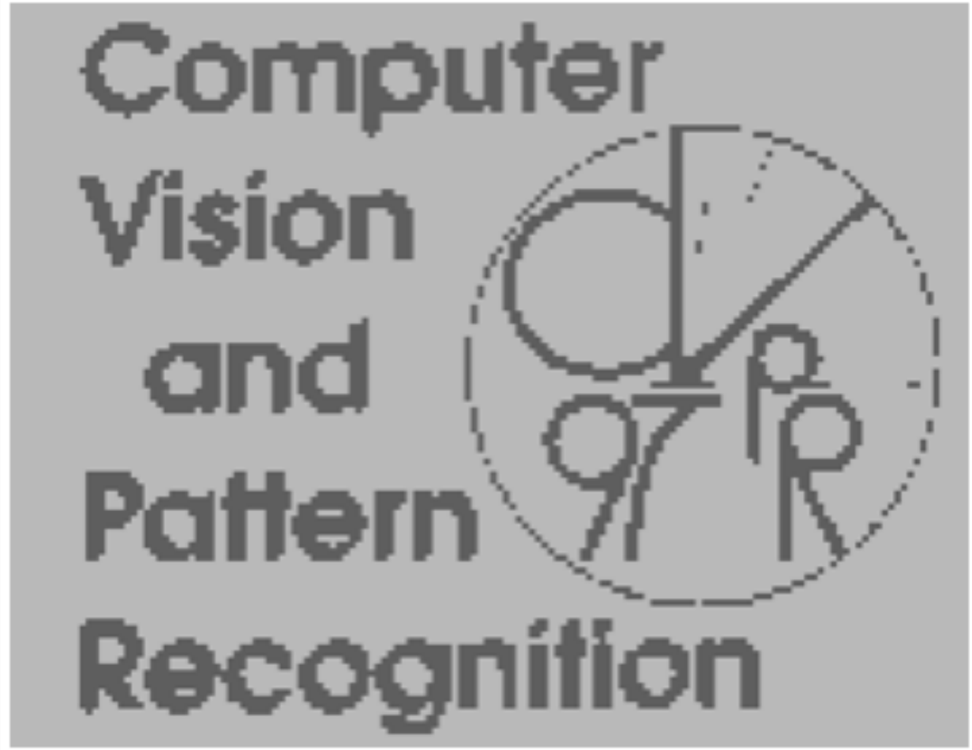


imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);

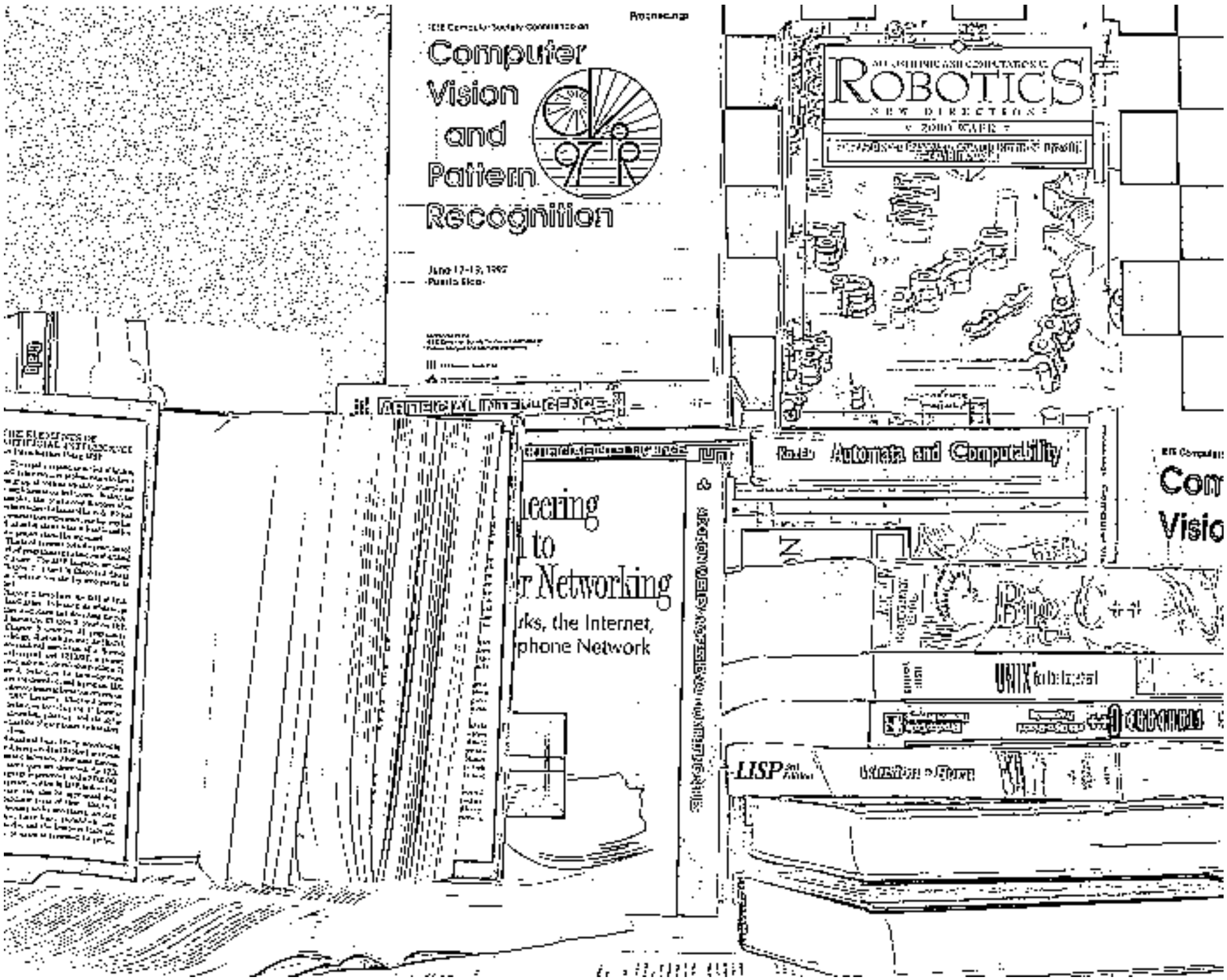


```
int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```
zoom in

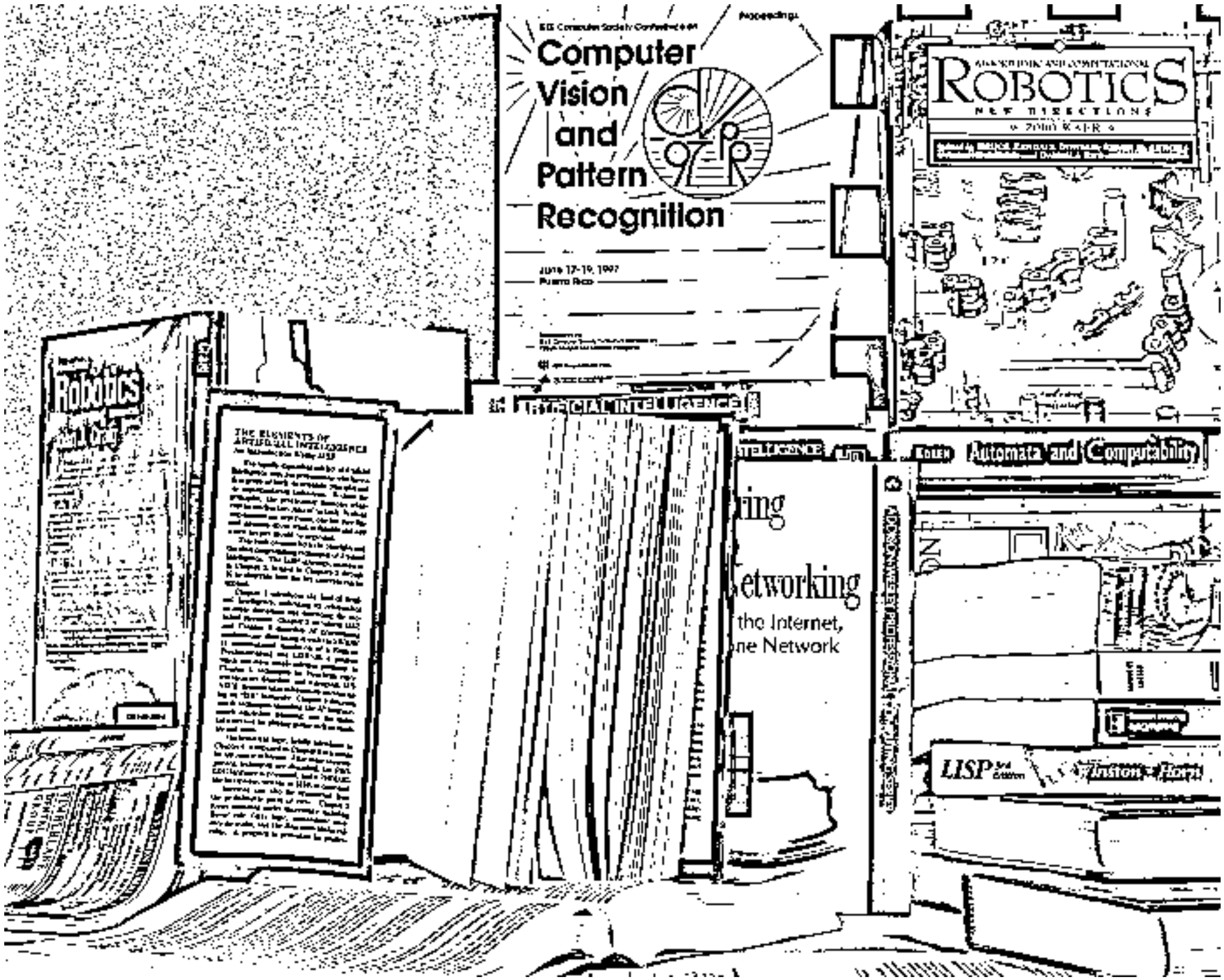




adaptive means, h=1

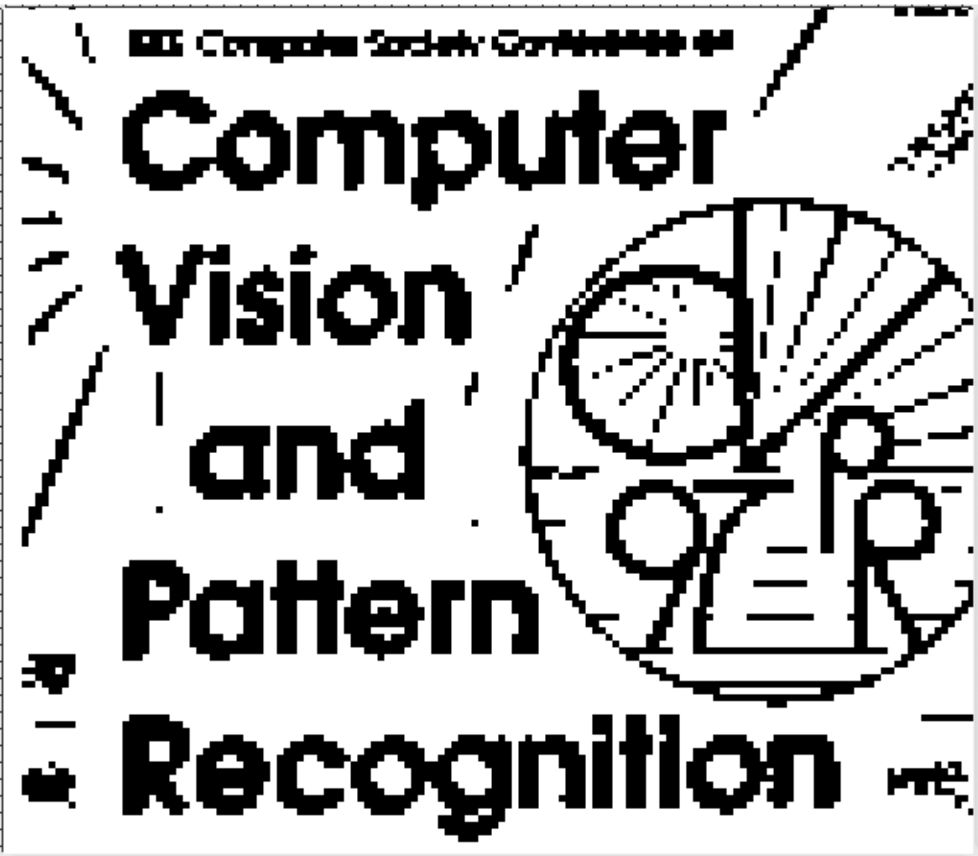


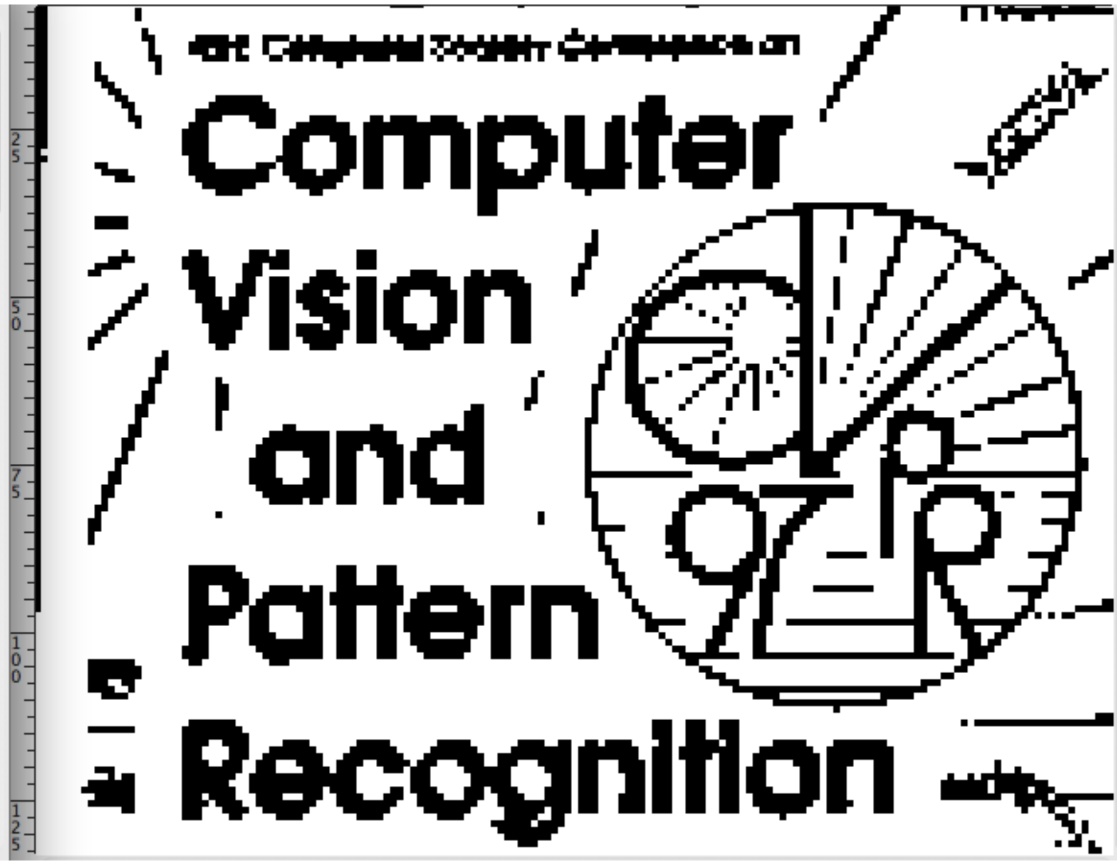
adaptive means, h=3



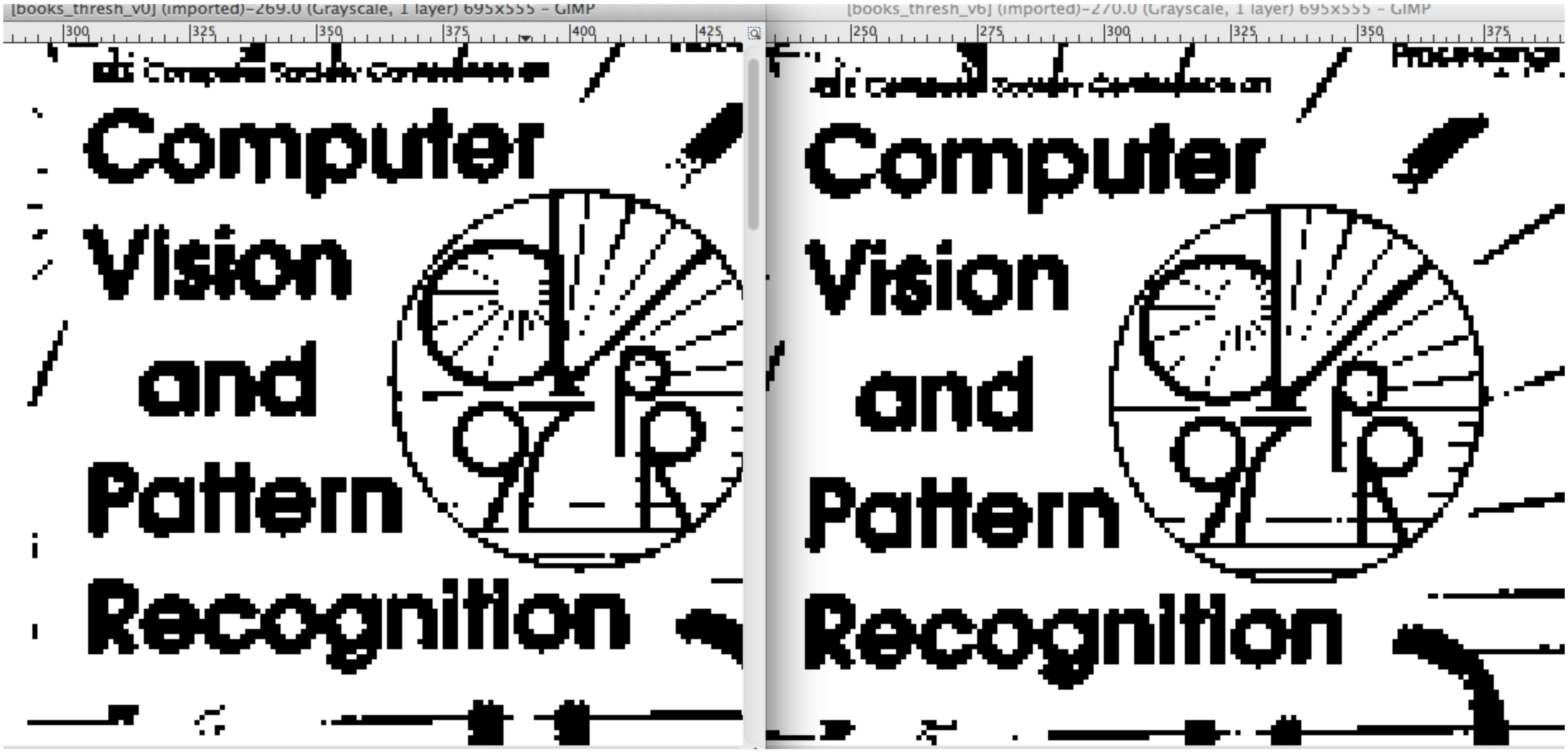
adaptive means, h=5

adaptive means, h=7

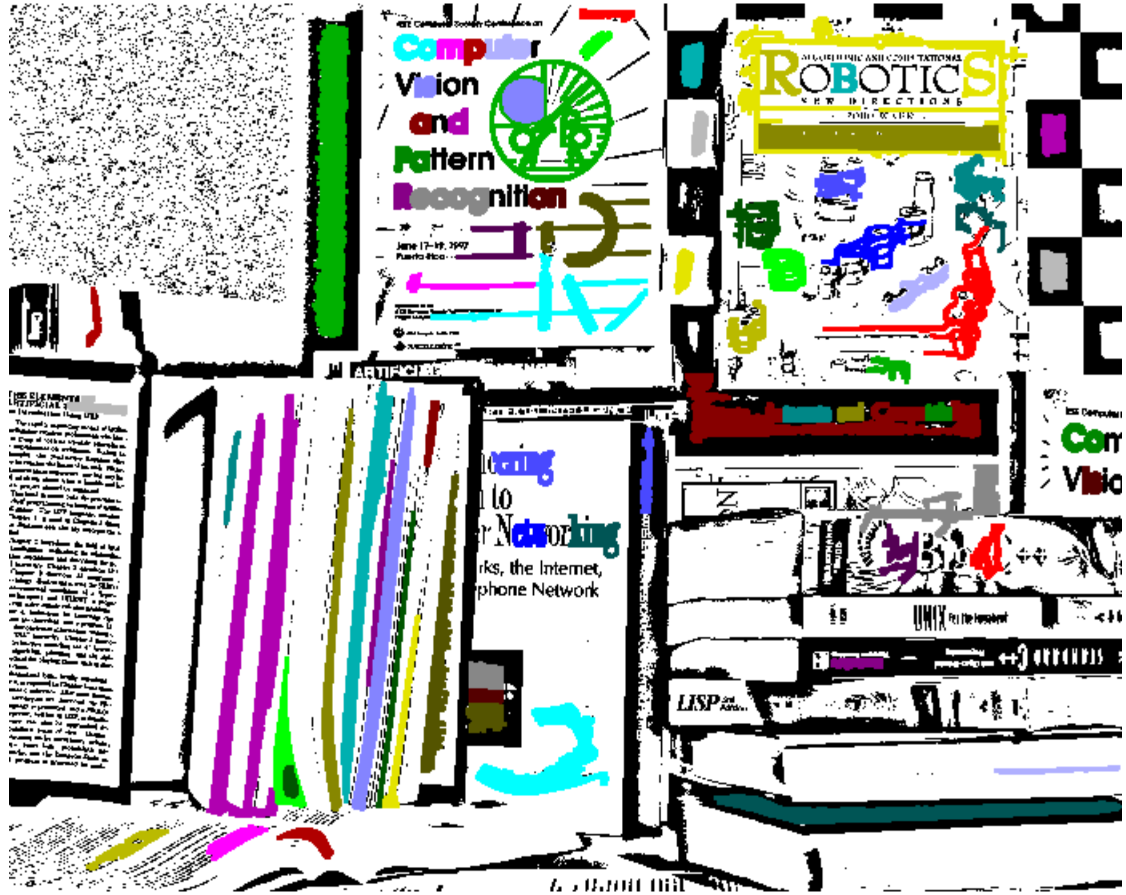




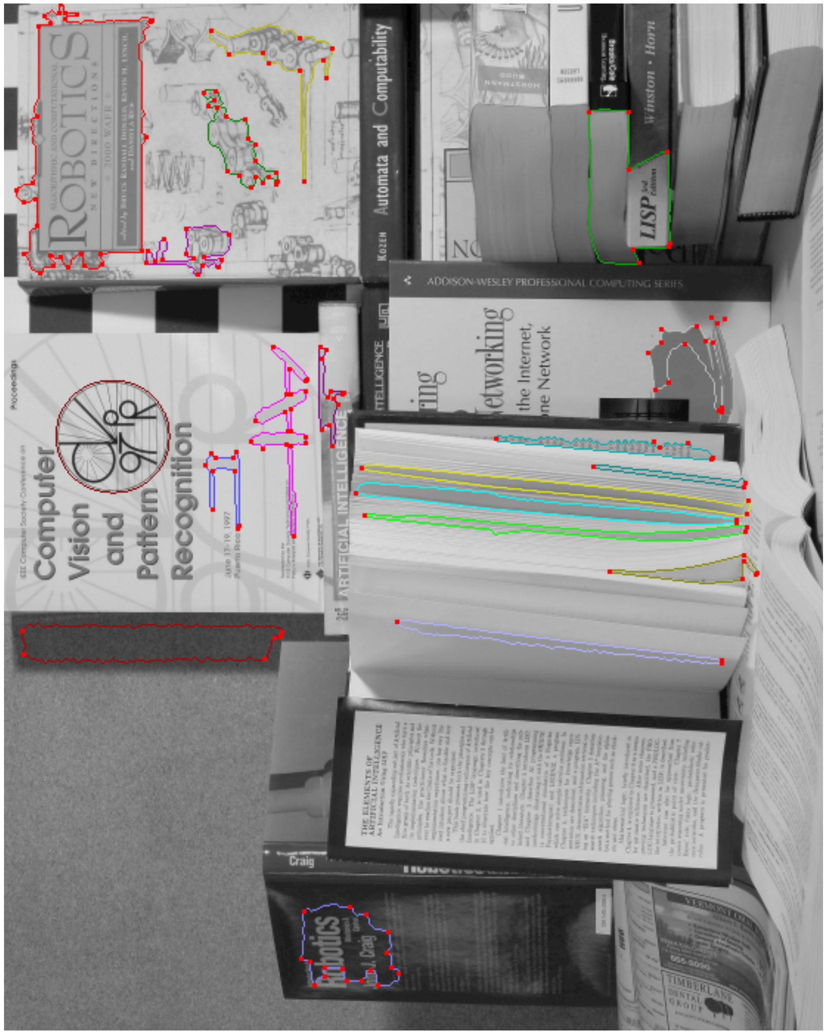
adaptive means, h=11 for adaptive to join close letters

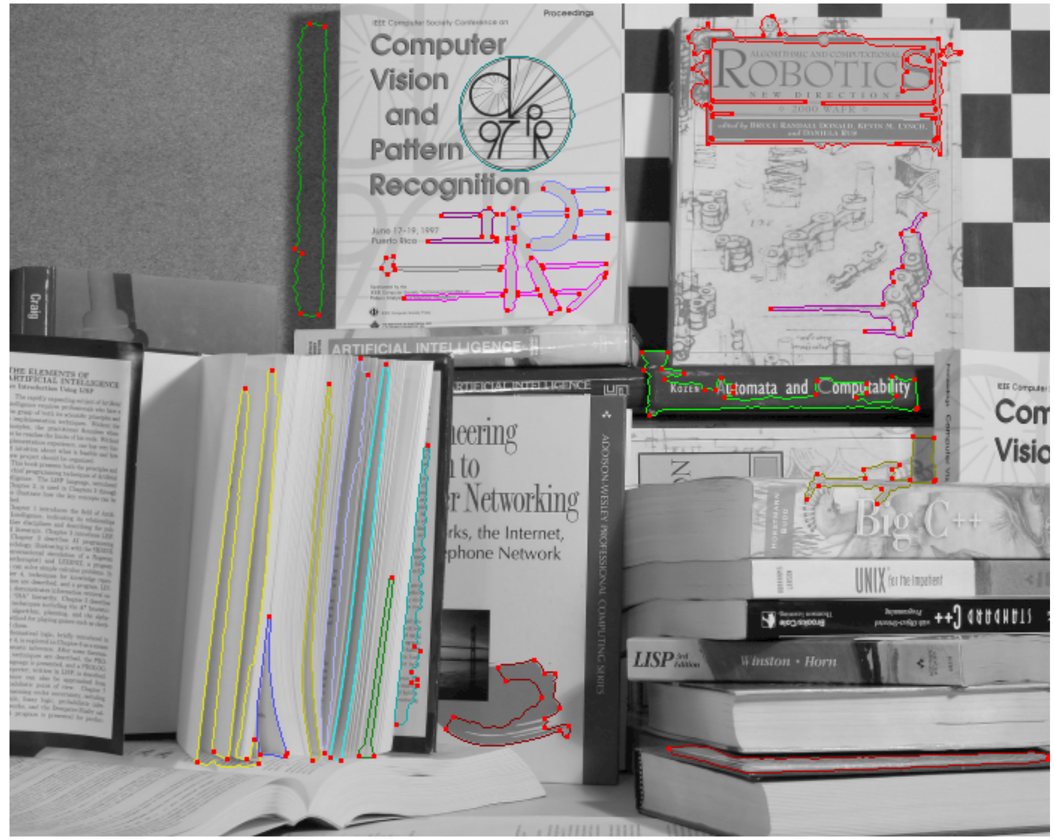


adaptive means, h=11 blobs (extr using scale calc code)



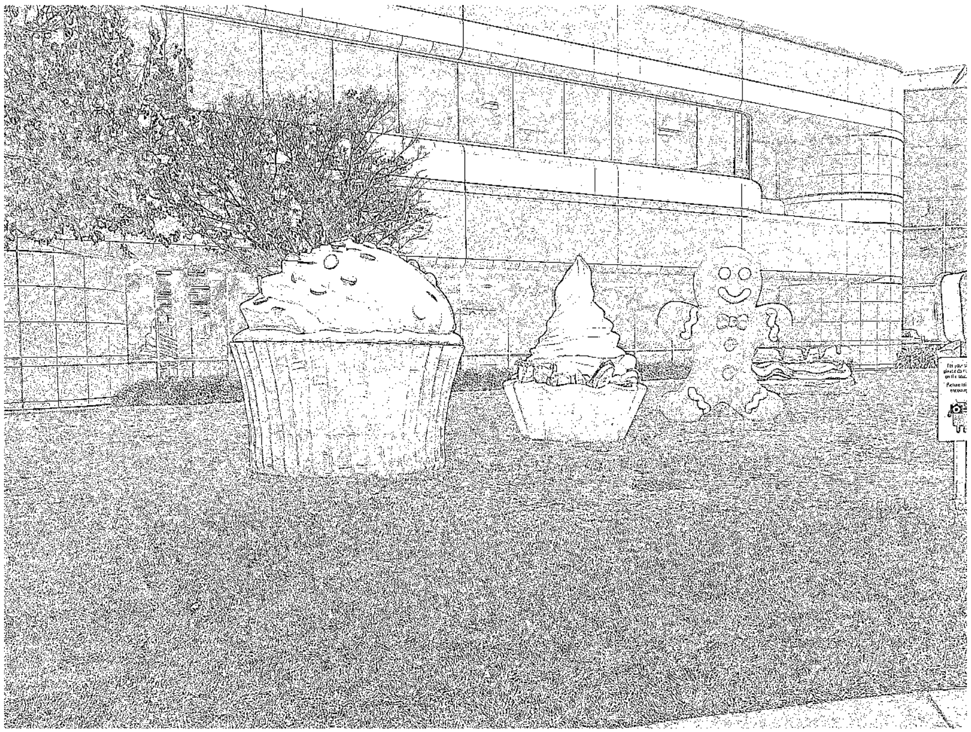
adaptive means, h=11 blob corners



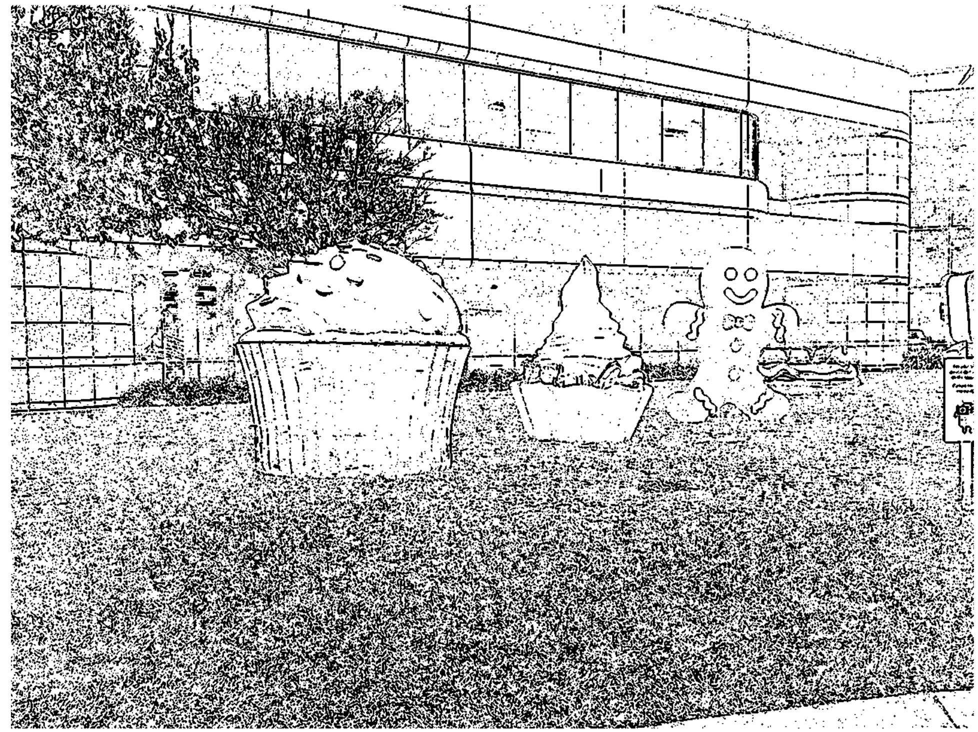


the letters are filtered out due to large nPerimeter compared to nInternal

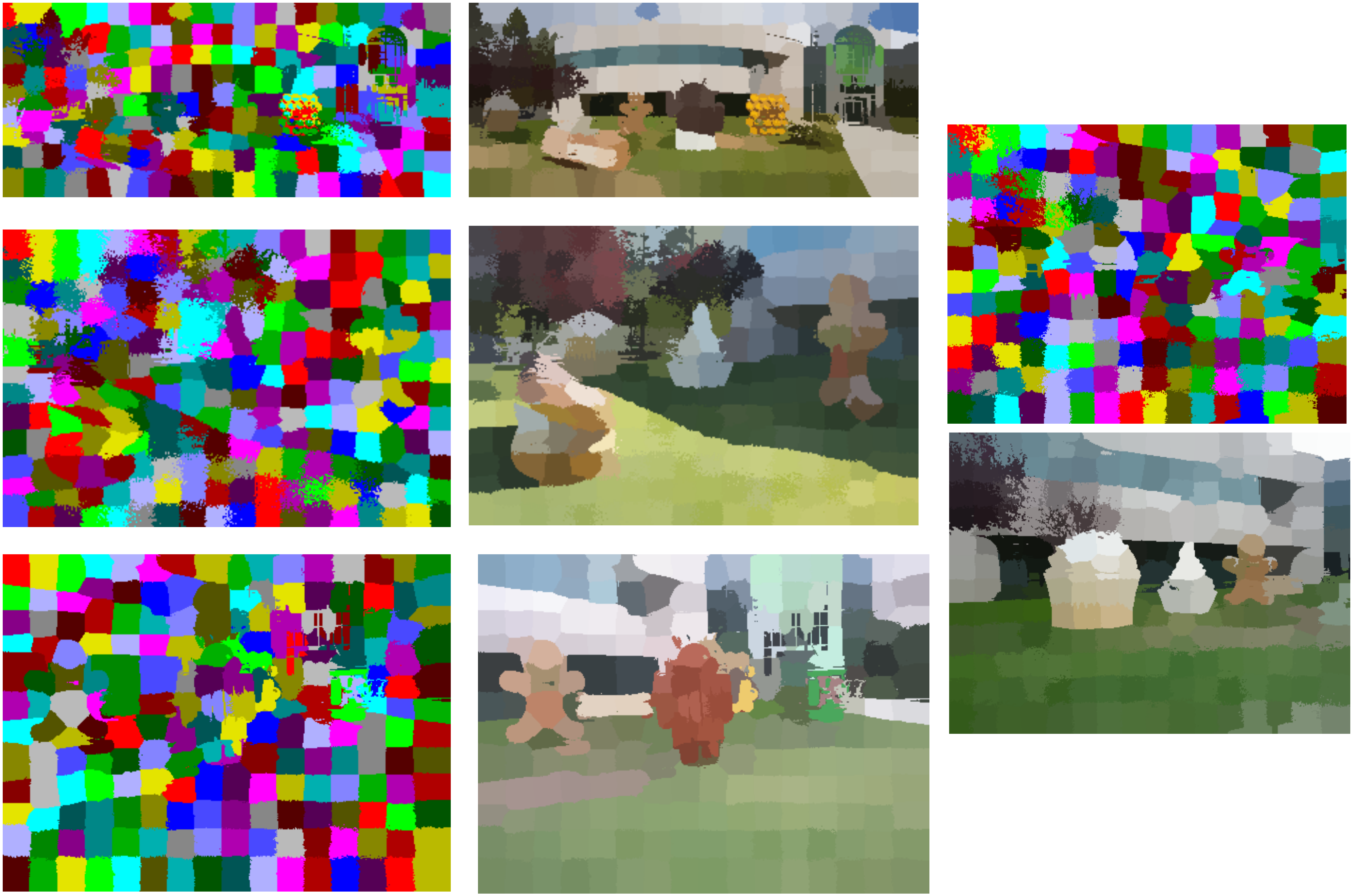
adaptive means, h=3 was better for letters as blobs adaptive means, h=1



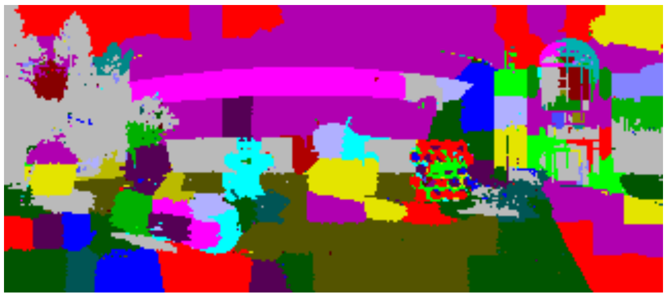
two median smooth with h=2

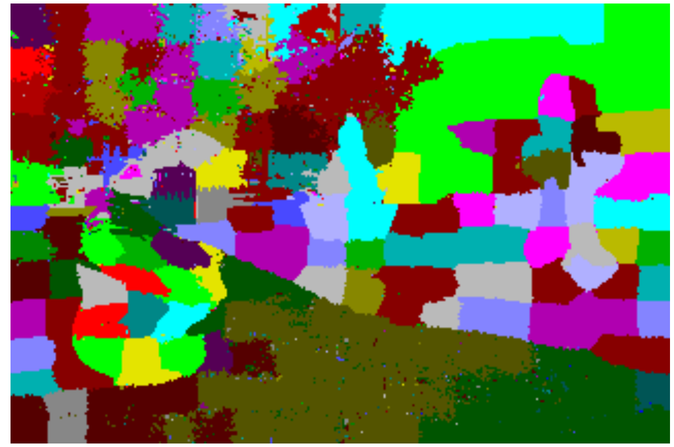


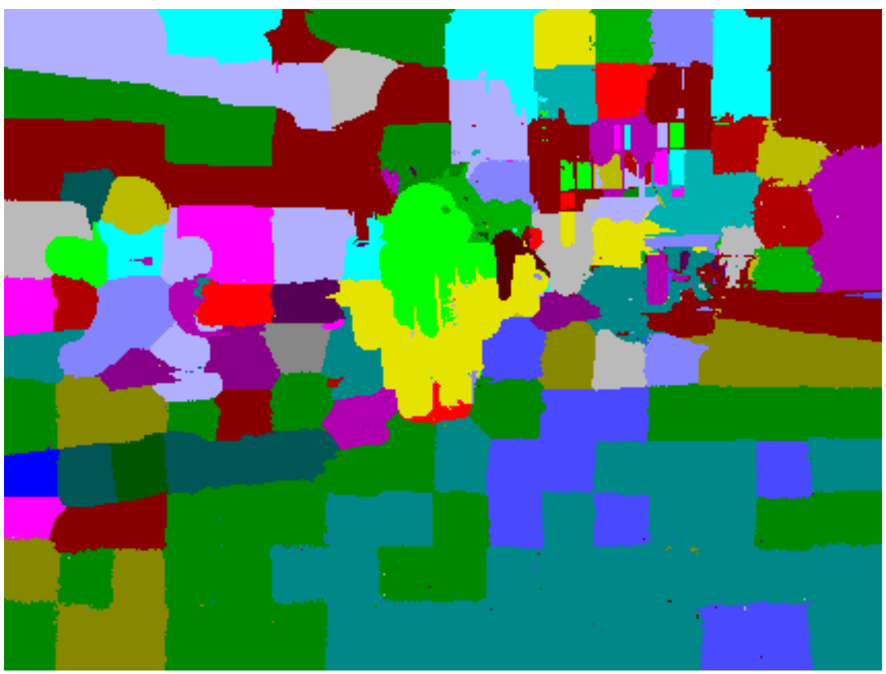
SLIC super pixels algorithm

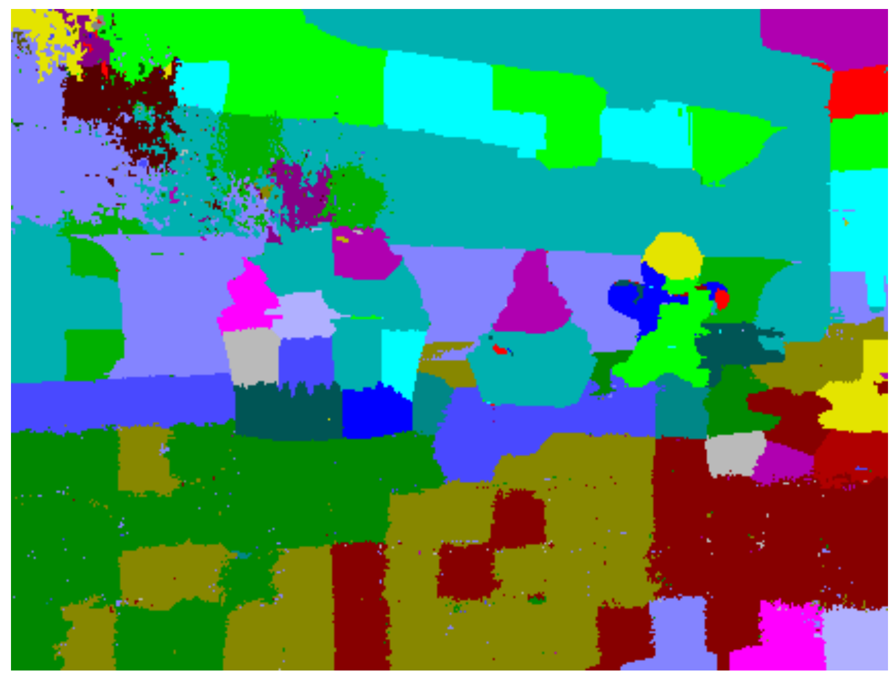


CIE XY Theta Clustering of the super pixels

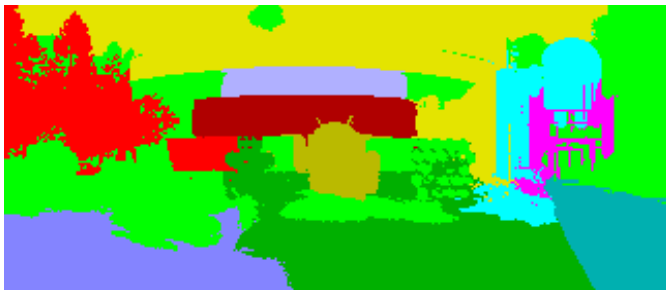


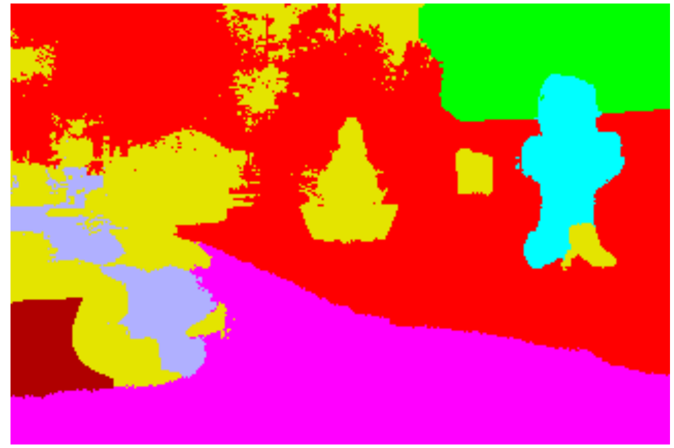


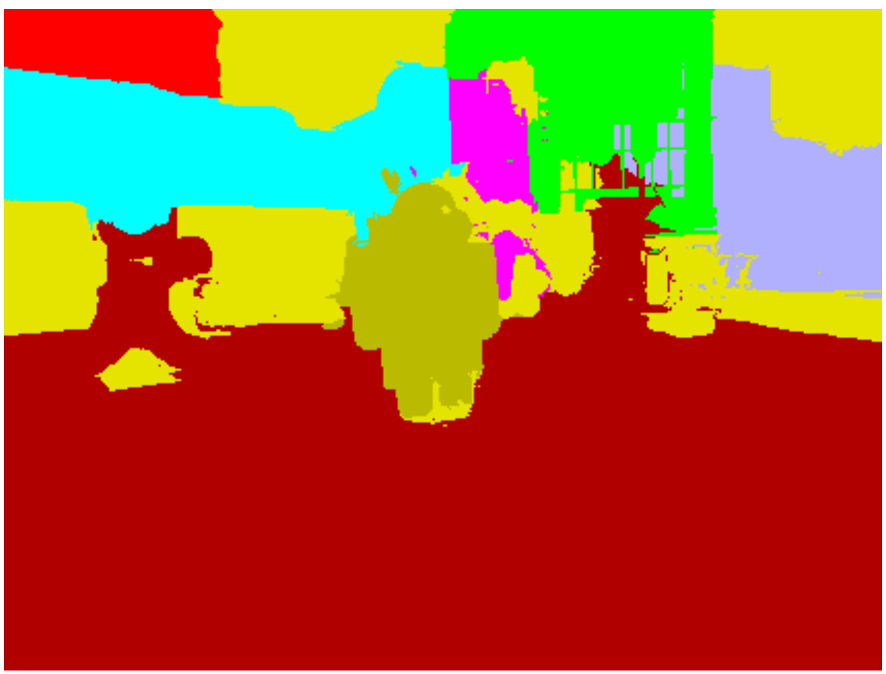


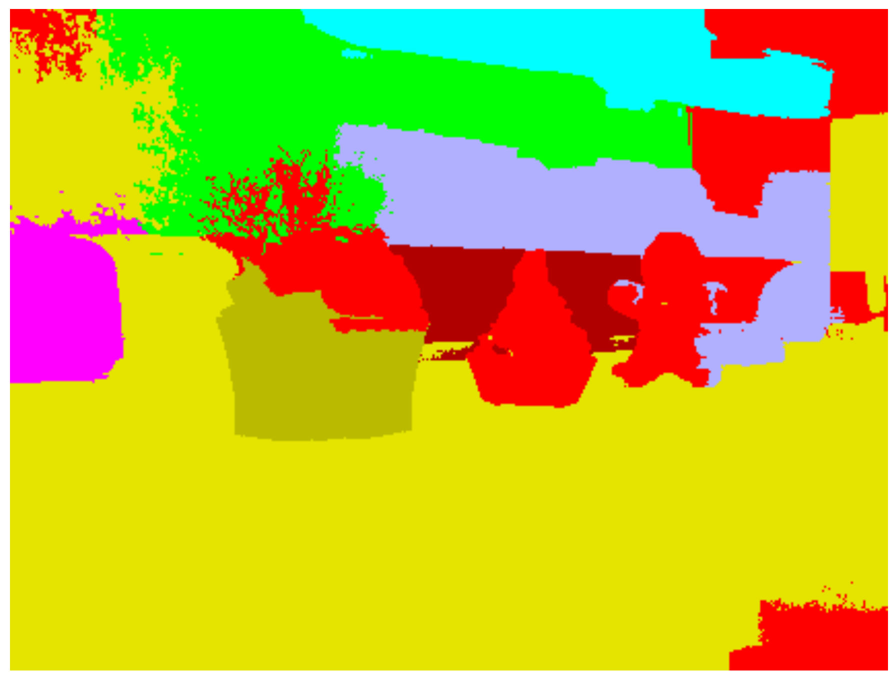


Normalized Cuts w/ HSV cs on the super pixels

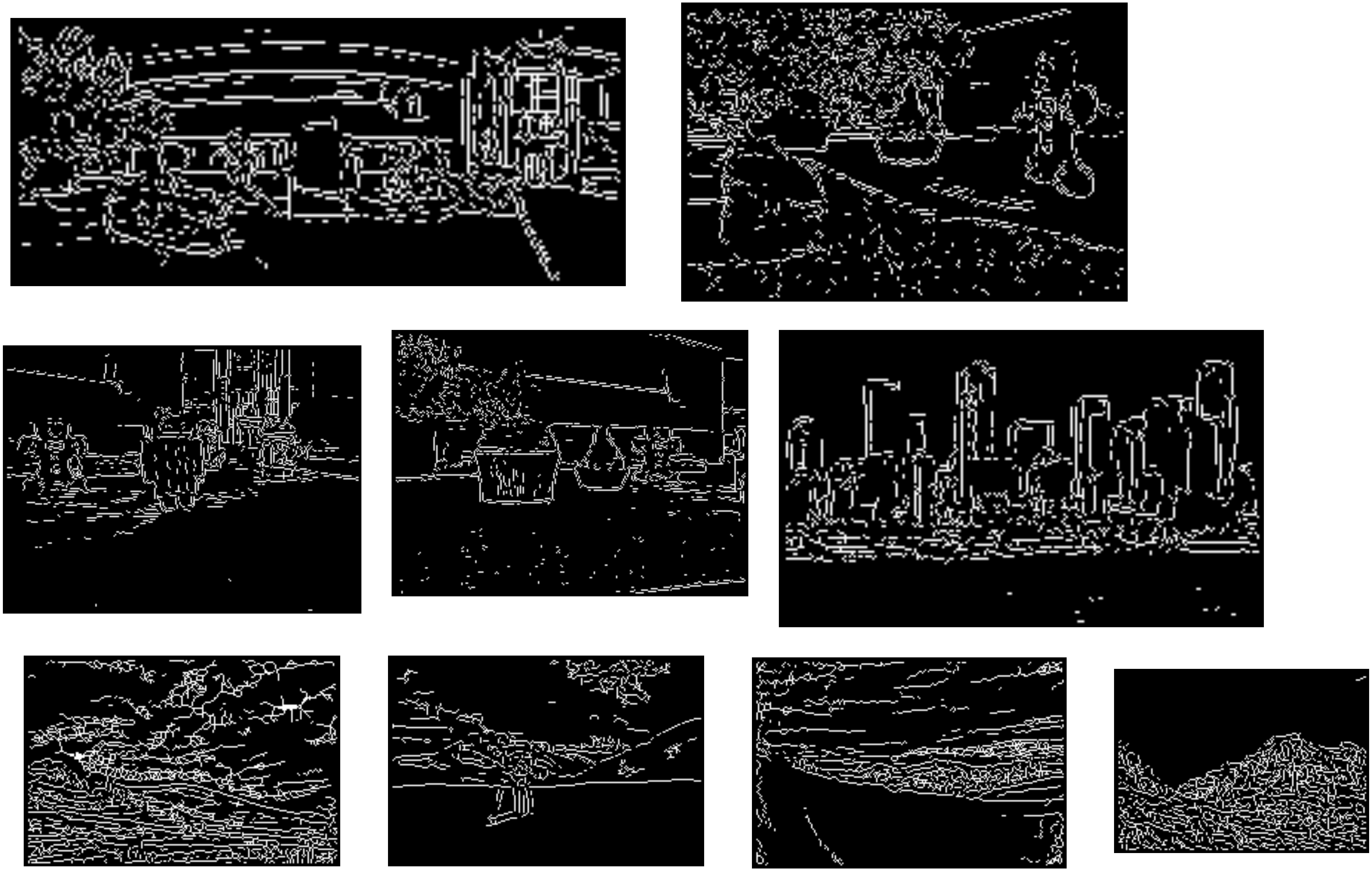




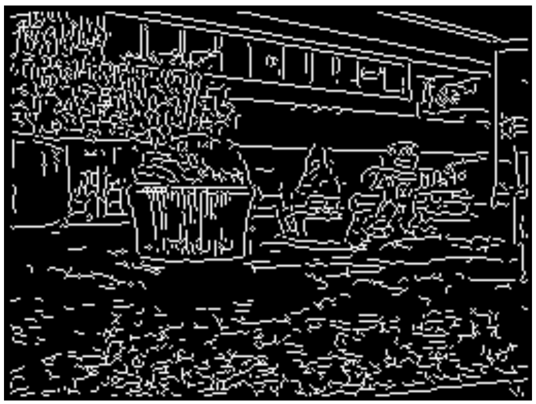


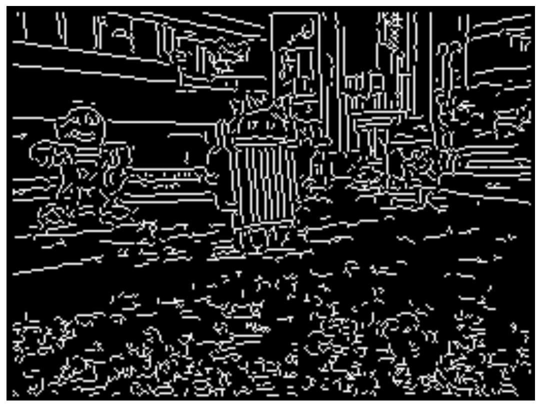


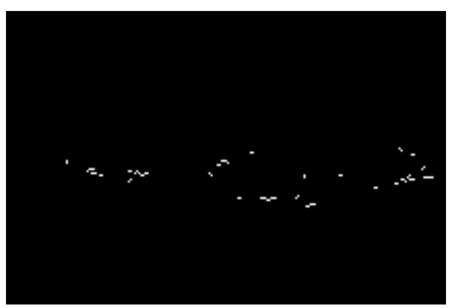
canny greyscale

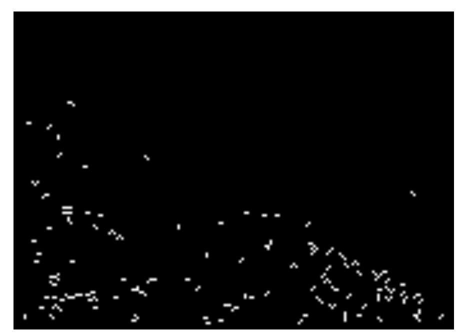


canny deltaE 2000





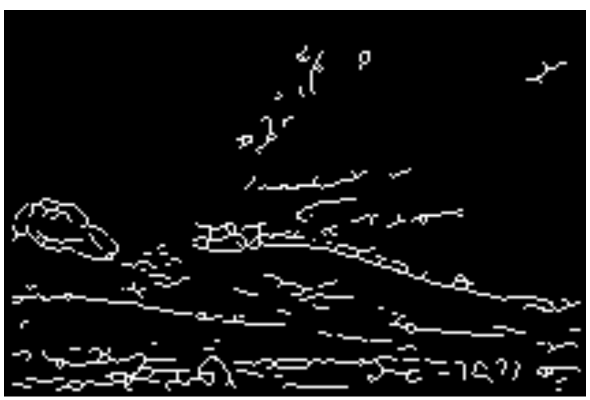




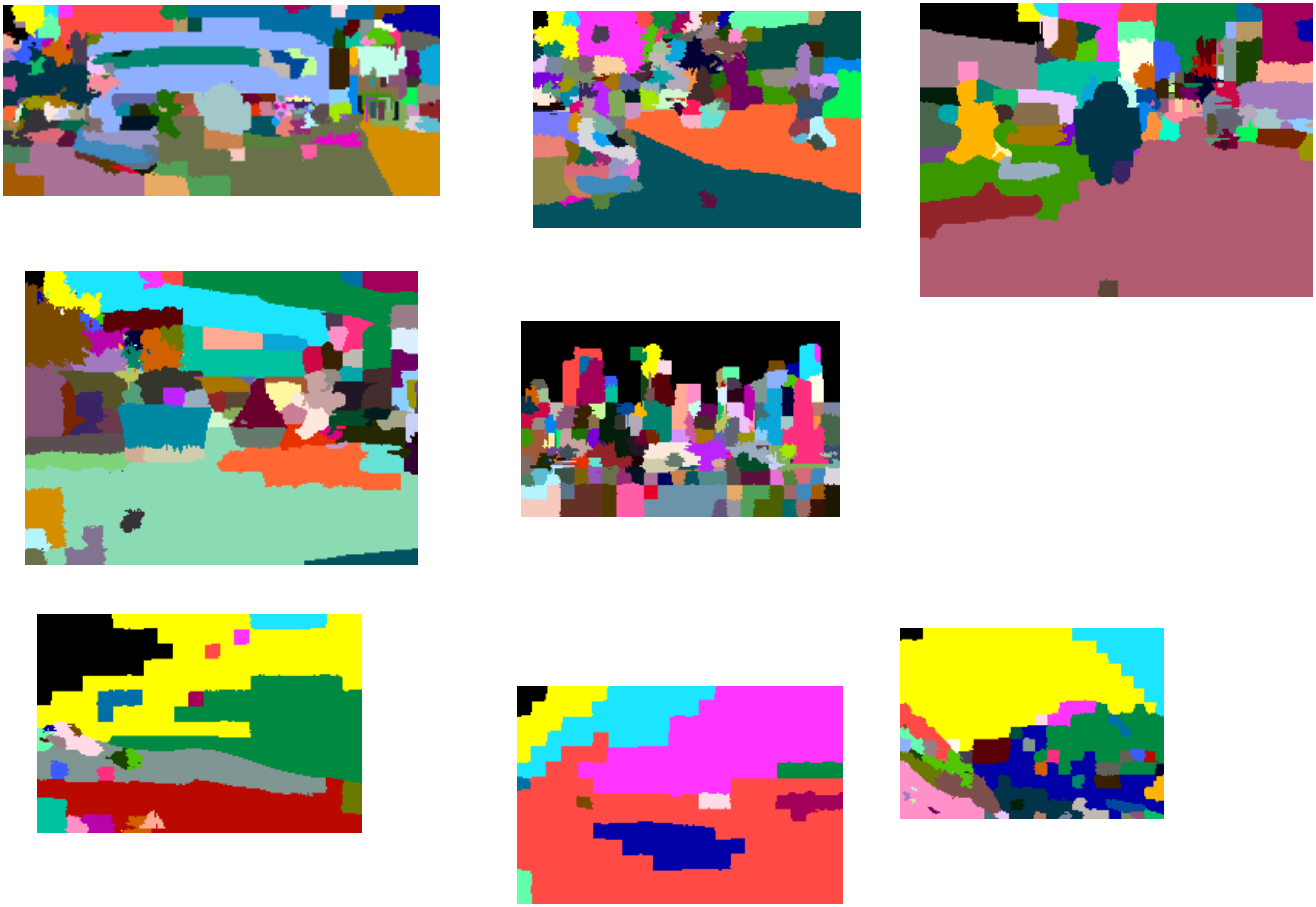
phase congruence





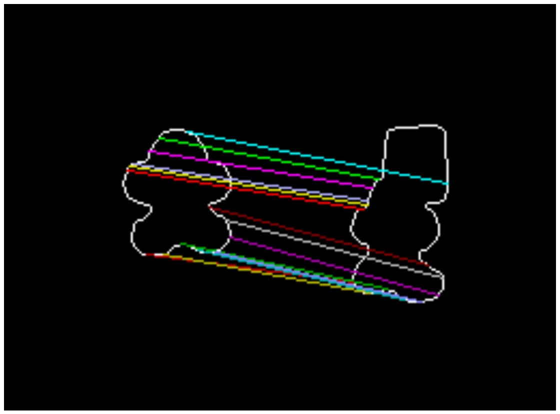



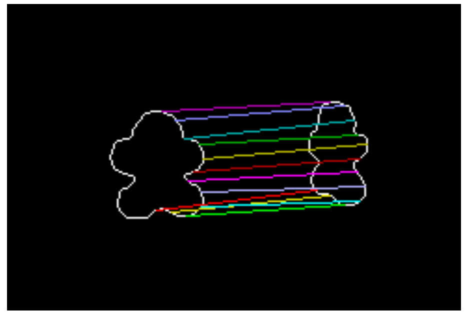
super pixels and moderate merging by HSV color

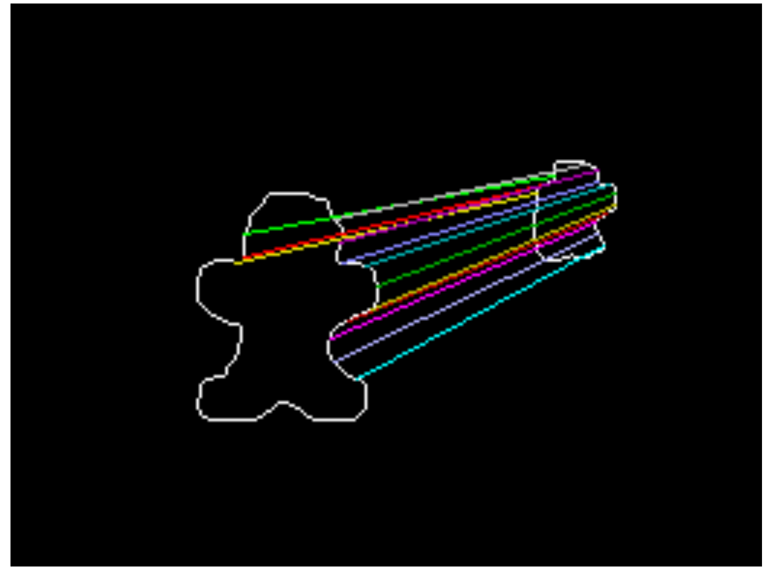


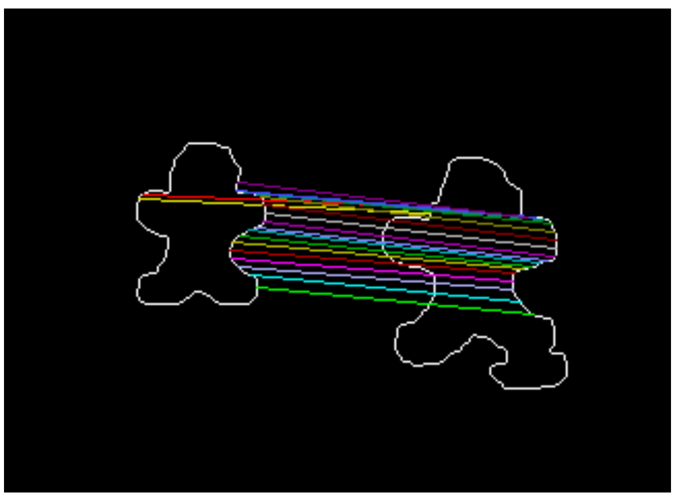
Partial Shape Matching of the gingerbread man

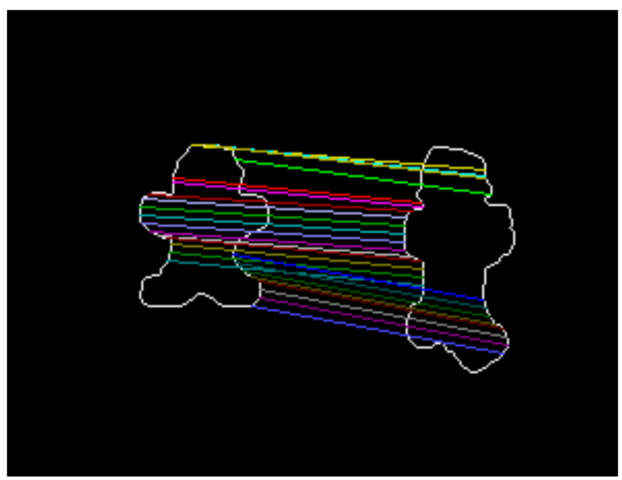
extract ordered borders of shapes, blur them by one or two sigma, then ordered bipartite matching algorithm using difference of chords as a descriptor.

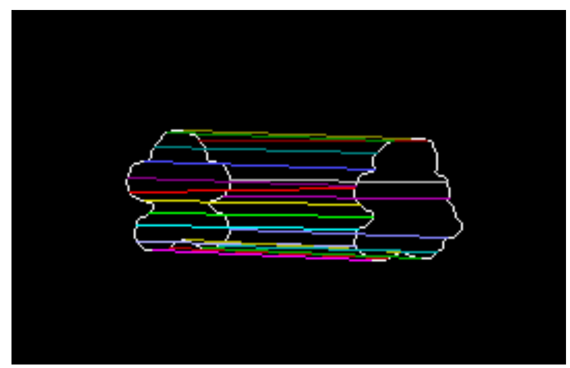












Shape Finding in over-segmented images

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

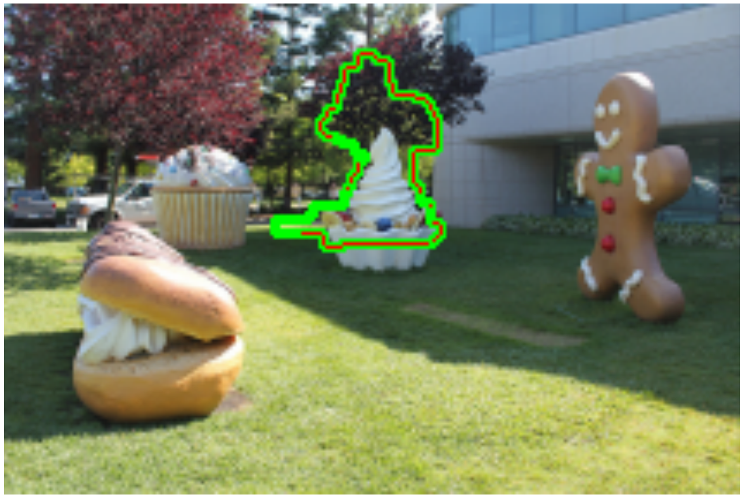
Finding the gingerbread man with just descriptors within the test images which have different lighting, poses and background did not work well *demonstrated in other notes) - could see that matching groups of points was necessary. Also found that if descriptors are used, they must exclude the background.

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.

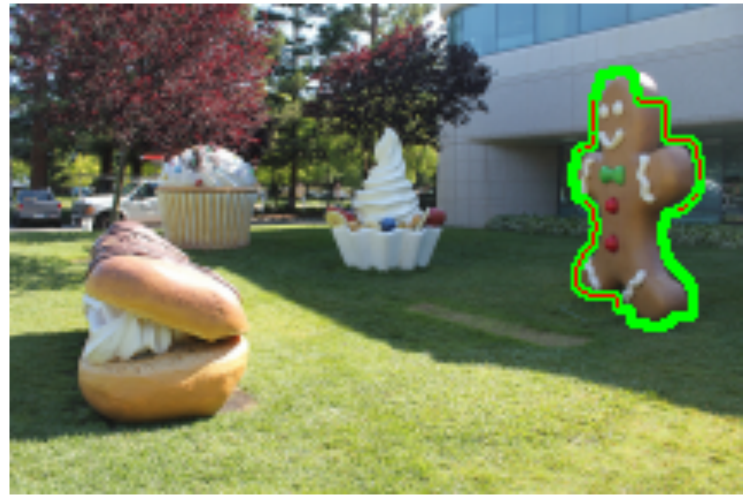
The search results are preliminary and still a work in progress, but can see that if more information about color or texture is used, the true match will be found as top match. NOTE that a histogram of which segmented cells are present in the top k answers might be a way to find the true match without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

top matches to template shape within a test image

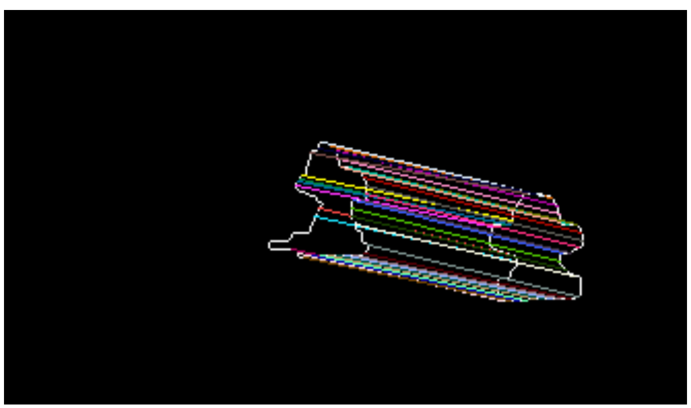
best match **and** **true match**

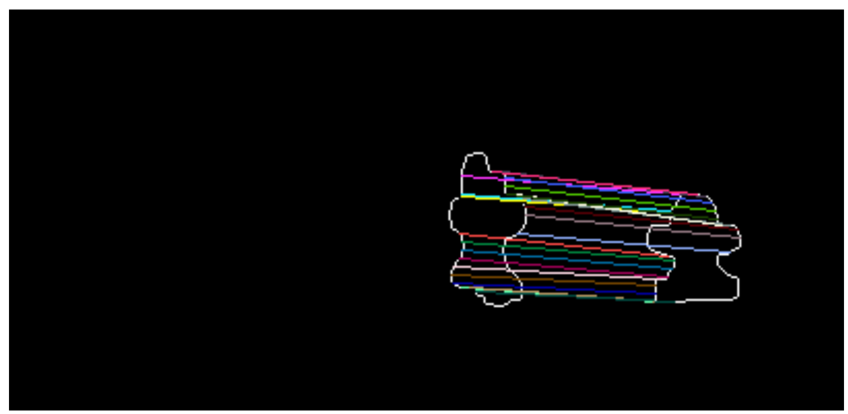


the min cost match to the template shape



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



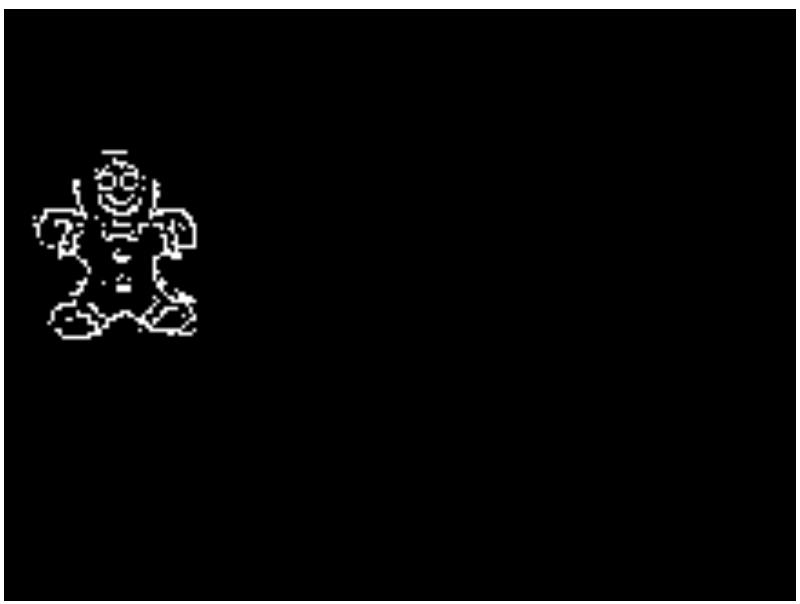


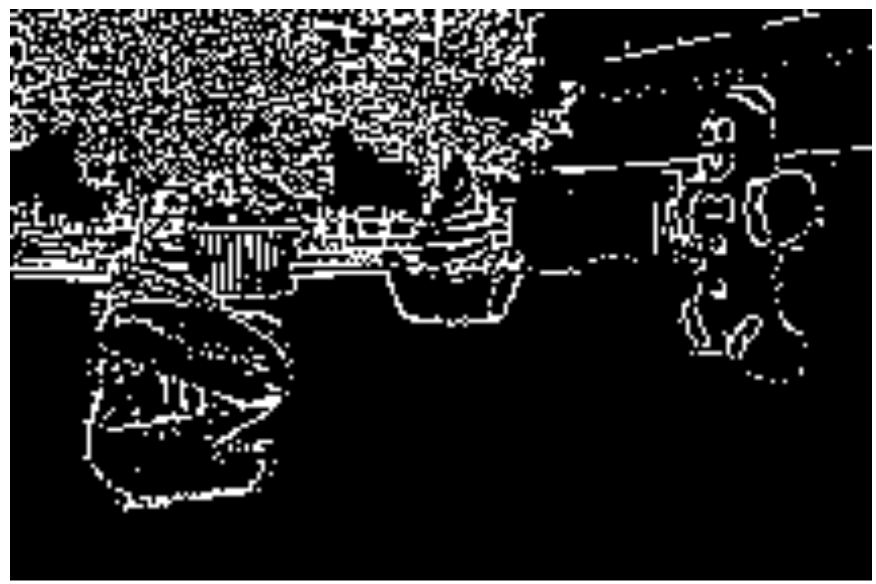
Shape Finding in over-segmented images (cont)

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

Shape Finding in over-segmented images (cont)

Adaptive means w/ half length=1 contains content internal to the segmented cells that could be used during cost calculation stages during aggregation in the shape finder.
Since the Euclidean transformation is known from the partial shape matching, the comparison of points of the adaptive means segments is possible within a range of projection variables.

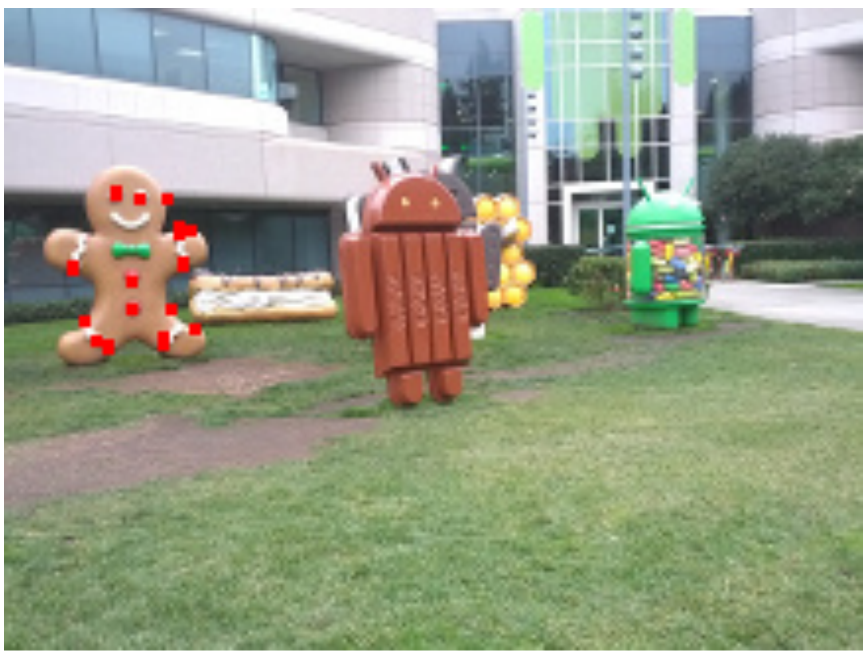


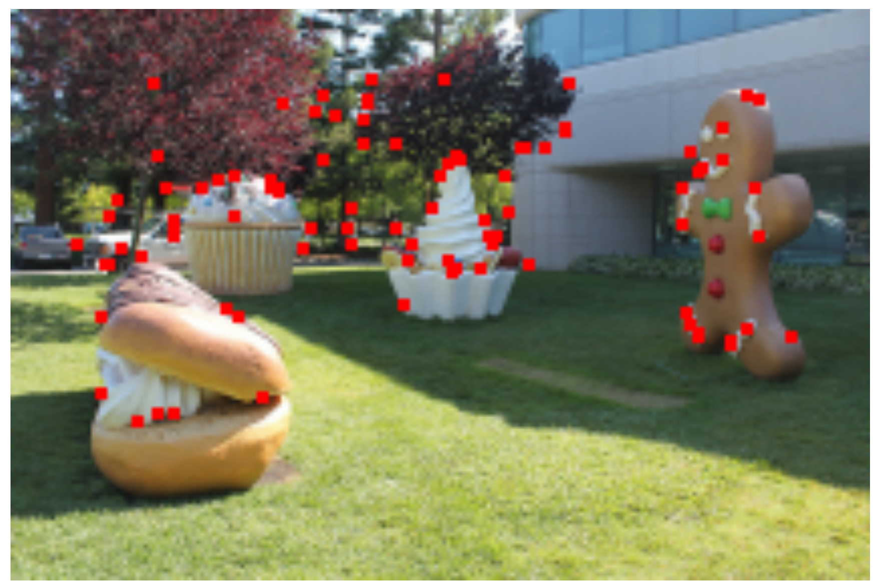


ORB key points and descriptors

skipping over more use of adaptive means for now, to implement the **ORB key points** and descriptors.

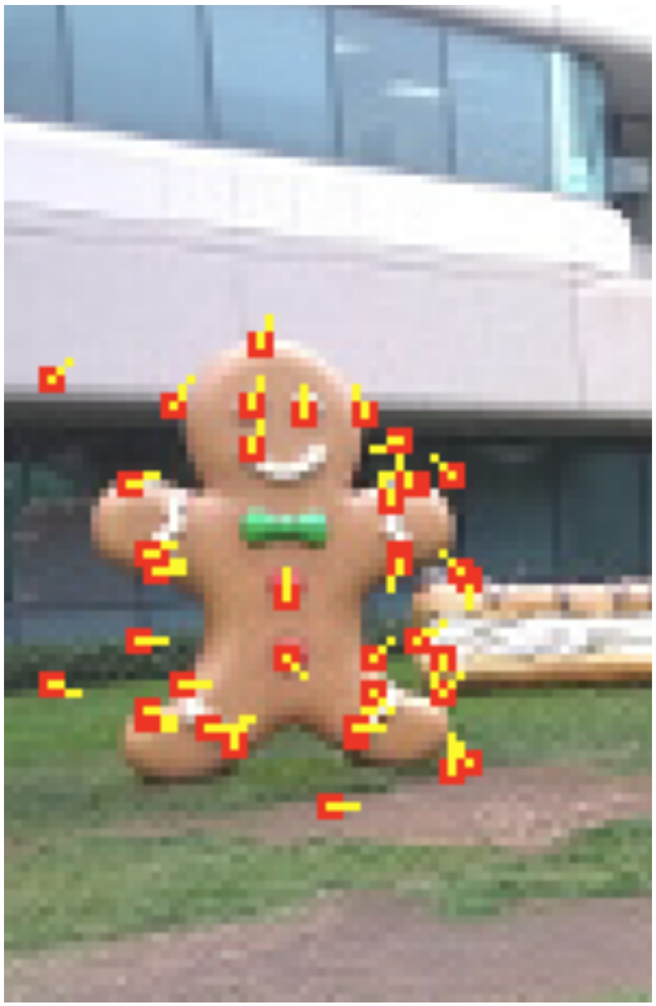
The key points for the template are at the left and on the right are key points filtered to those present in the segmented cells of the image to search, those cells were filtered by color histogram intersections with the template gingerbread man.

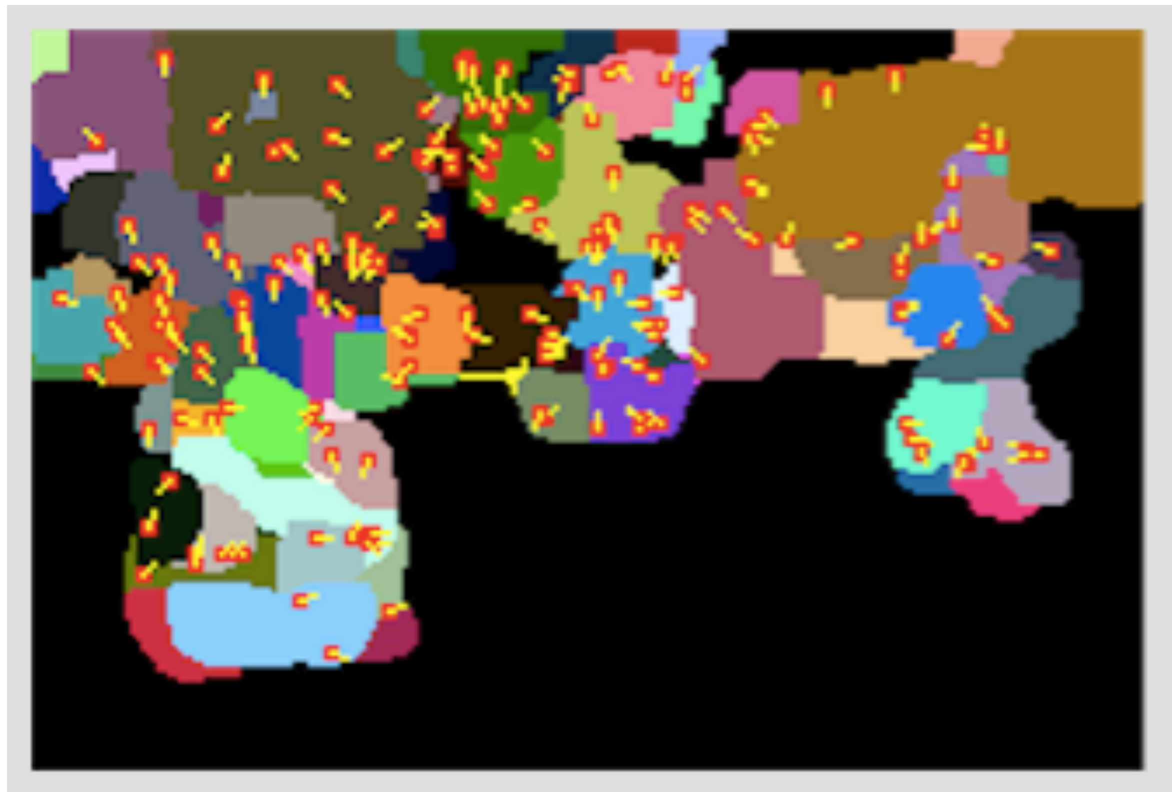




ORB key points and descriptors (cont.)

The orientation angles of the ORB detector are plotted as extended yellow lines away from the red key points.

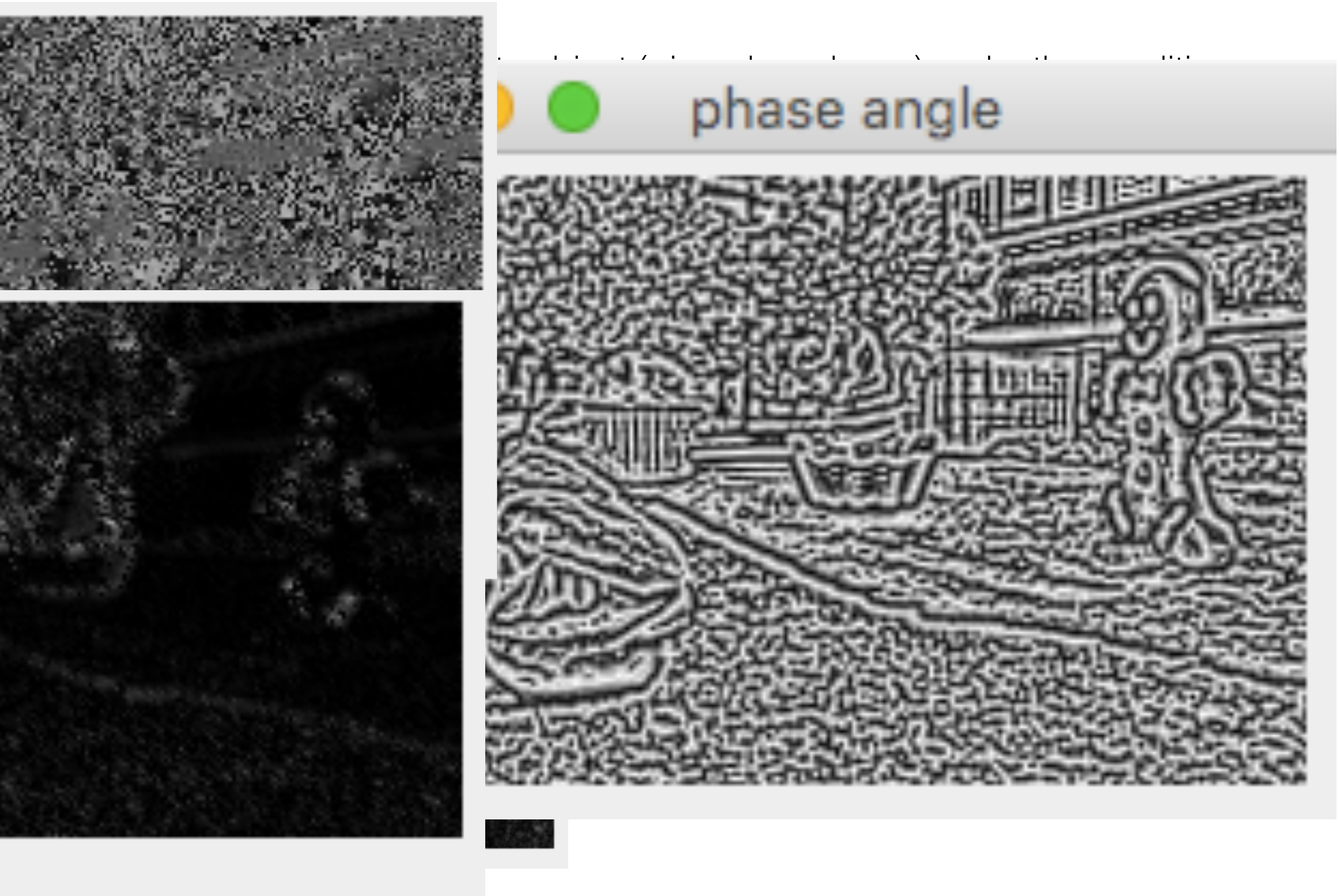




ORB key points and descriptors (cont.)

Added more key points to the **ORB detector**.   The new points are the maxima of curvature and are many in number.

A quick look at matching the template descriptors to all keypoints in the search image shows that a solution is now possible. Only the gbman matches are shown. .

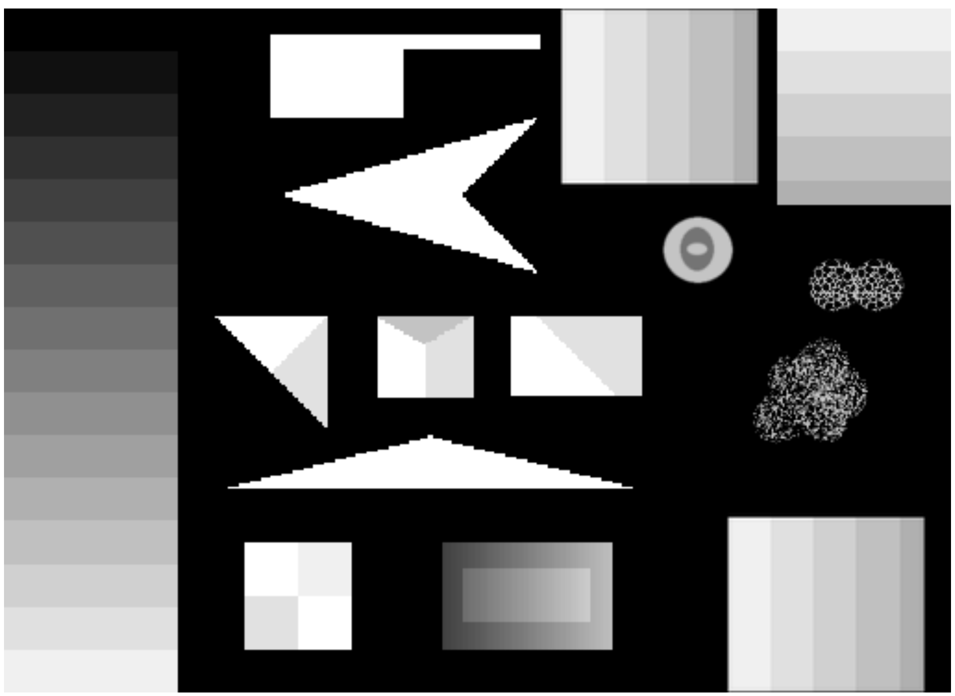

phase angle

texture

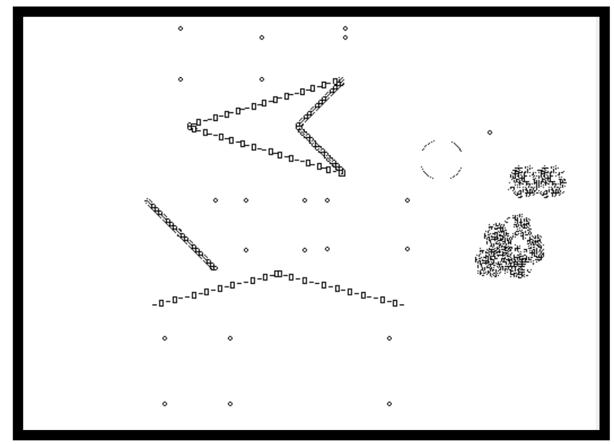
The highly textured regions such as vegetation produce many key points and those need to be filtered out for this case.
The Law's 1980 texture transforms (combinations of gaussian derivative convolutions) show that R5R5 may be a good way to find regions of high spatial and intensity variability, that is high density changes.

E5 = [-1 -2 0 2 1] 1st deriv, sigma=1
R5 ripple = [1 -4 6 -4 1] 3rd deriv

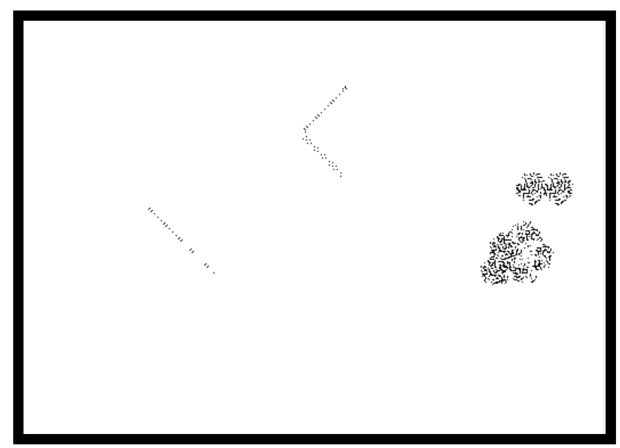
original image



E5E5 then adap med



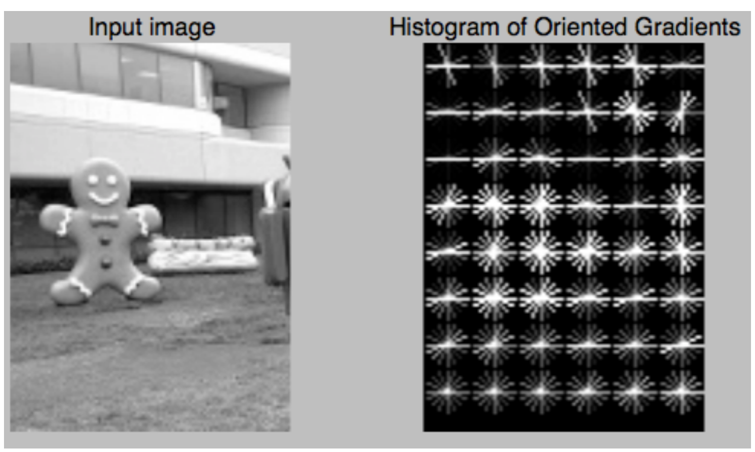
R5R5 then adap med

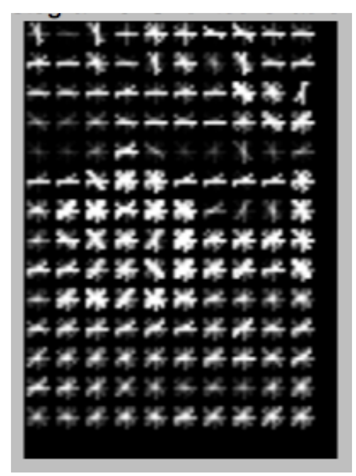


ORB key points and descriptors

Within the goal of matching the gingerbread man in different poses, lighting, and fore/background, have added an implementation of ORB key points and descriptors (adapted from scipy implementation).

The android statue test images are reduced in size to max length of 256. ORB descr are impl w/ fixed pair sampling relative to the key point orientation angle. The maximum dimension of that ORB descriptor aperture is 16 pixels.
False positives were found as best matches.
Can see in the images below that for these test conditions, need a smaller aperture than 16 to resolve the unique characteristics of the object.
Below, are snapshots of HOG results using 16 then 9 as visualization.

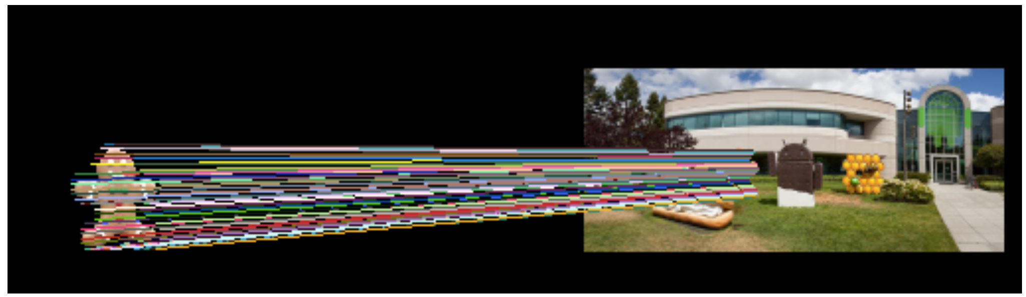




making a smaller ORB descriptor…

ORB key points and descriptors (cont.)

Found that for these small object cases, it was better to not use the ORB key points and descriptors, but to instead use CIELAB polar theta histogram intersections, partial shape matching, and evaluation of nearest transformed matching points learned from the shape matching transformation.



For the other images, ORB descriptor matching with a pairwise euclidean matching of same label points worked.