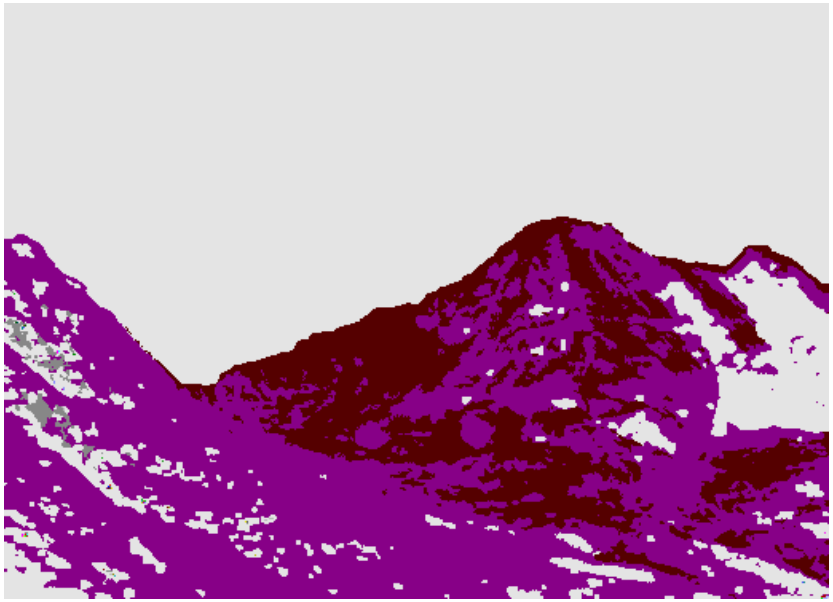
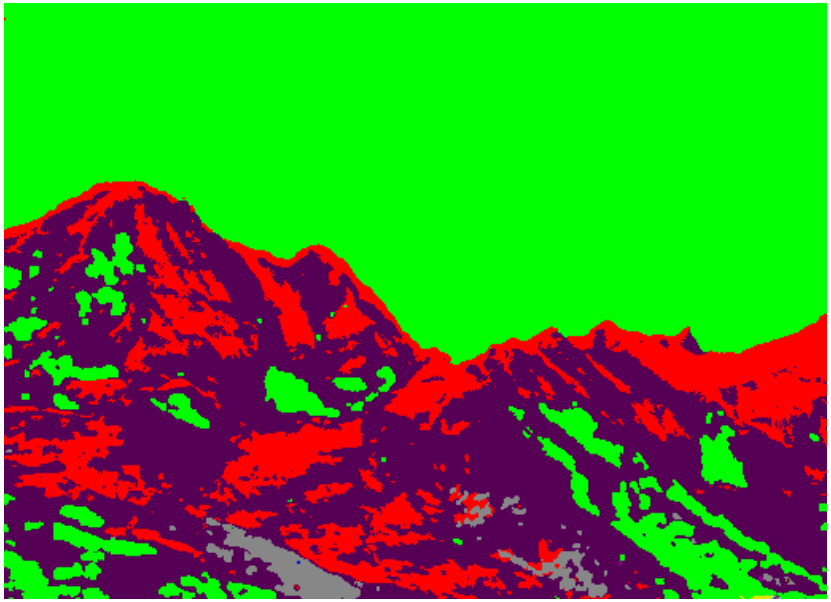
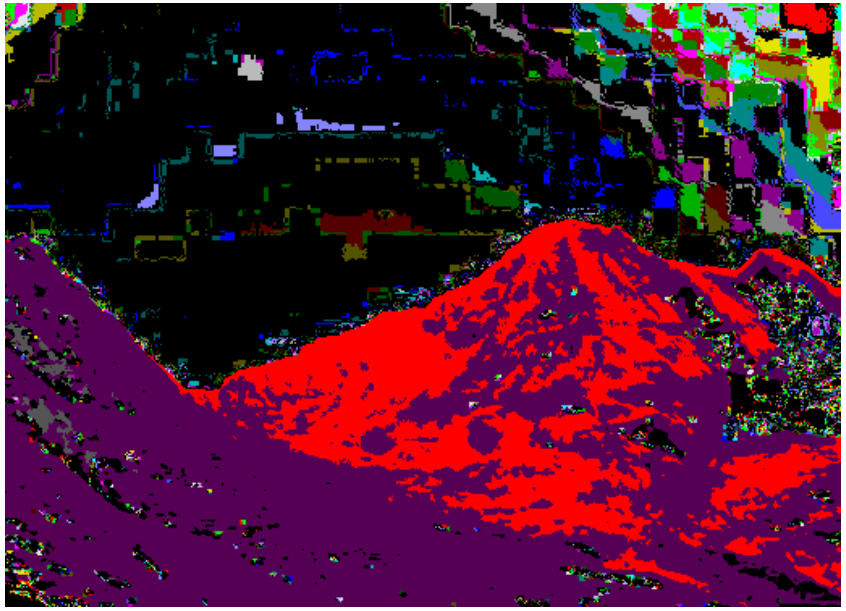
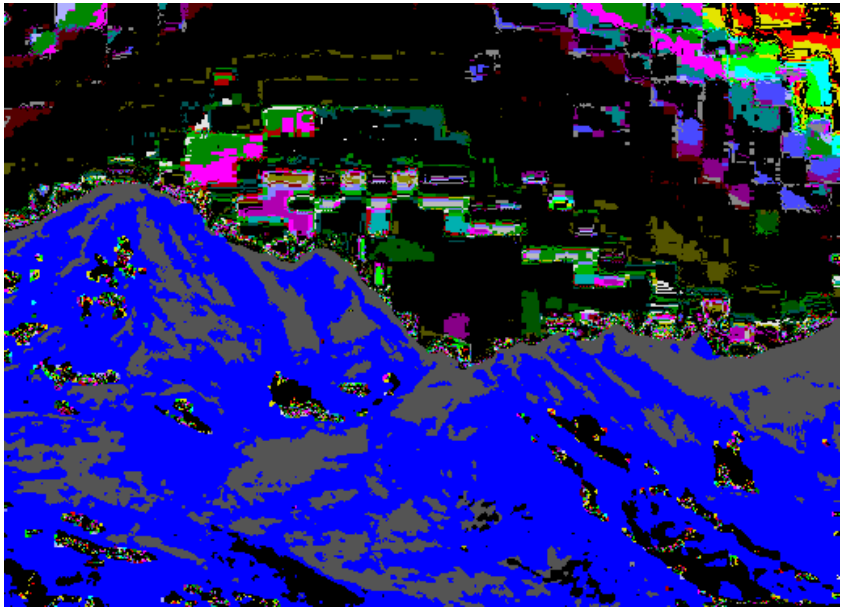
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);



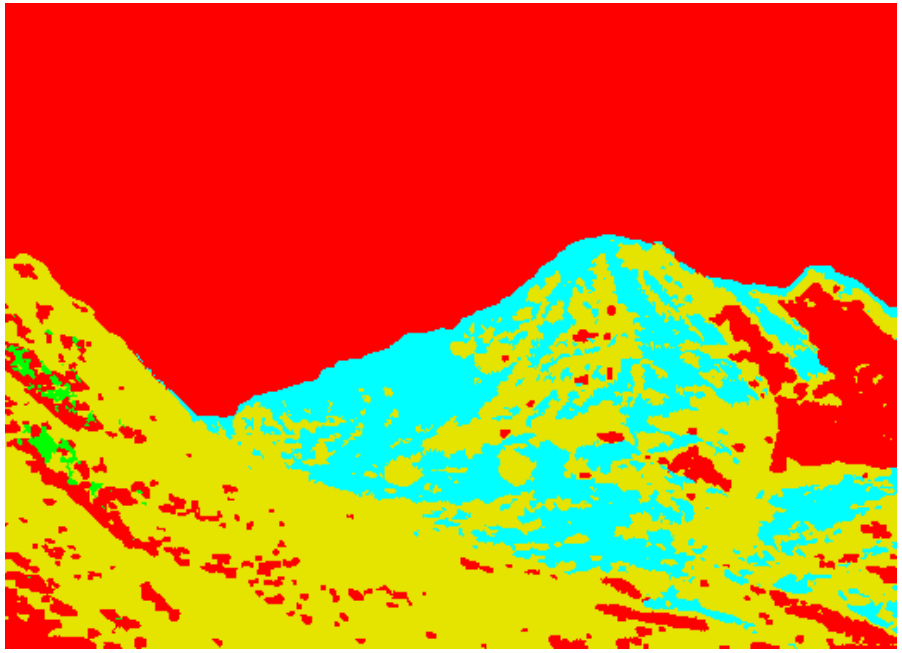


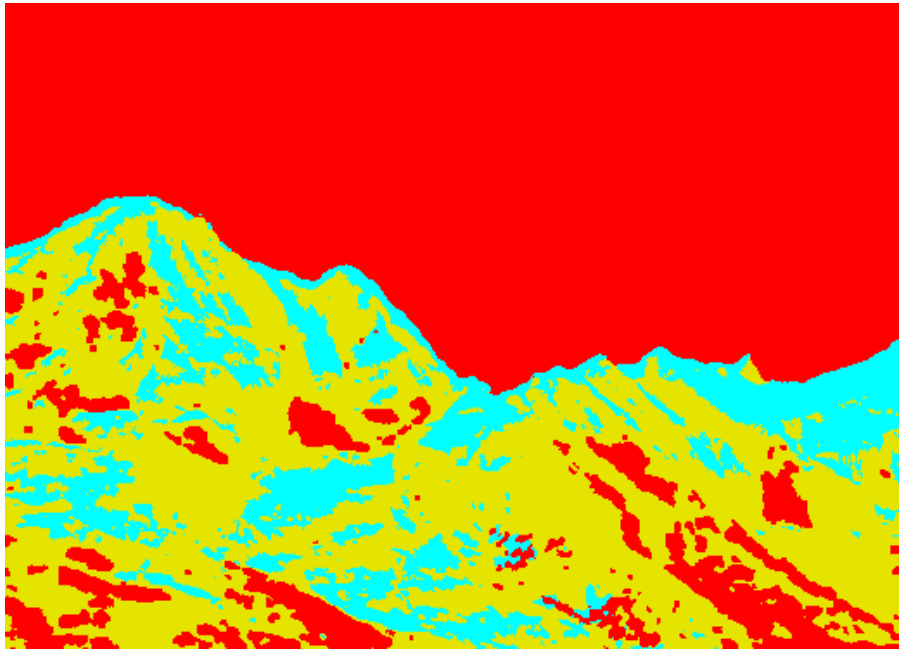
imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);



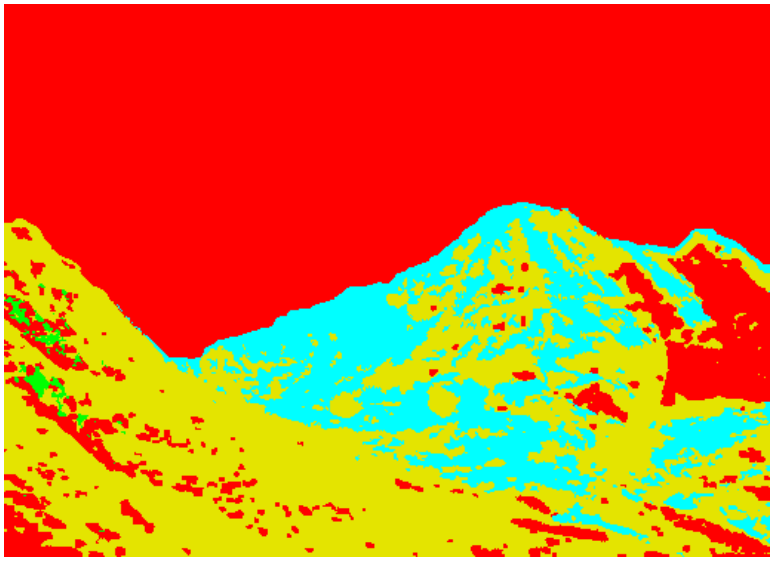


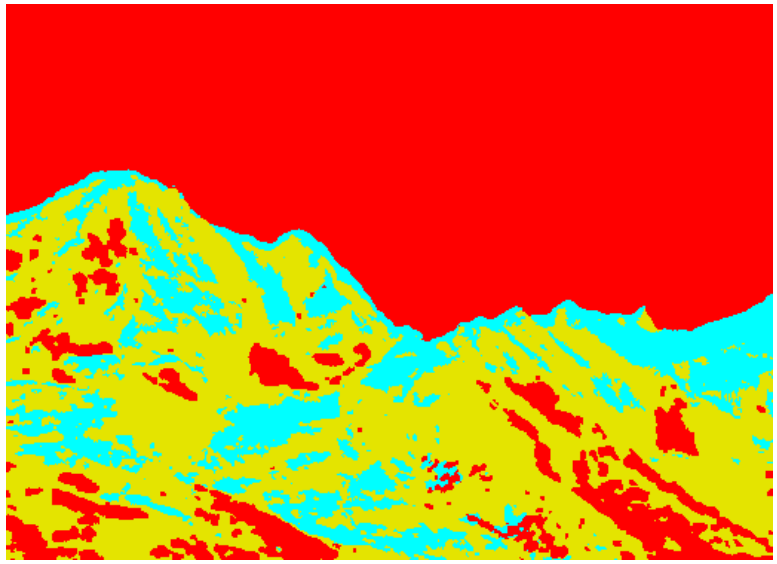
```
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);
```





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





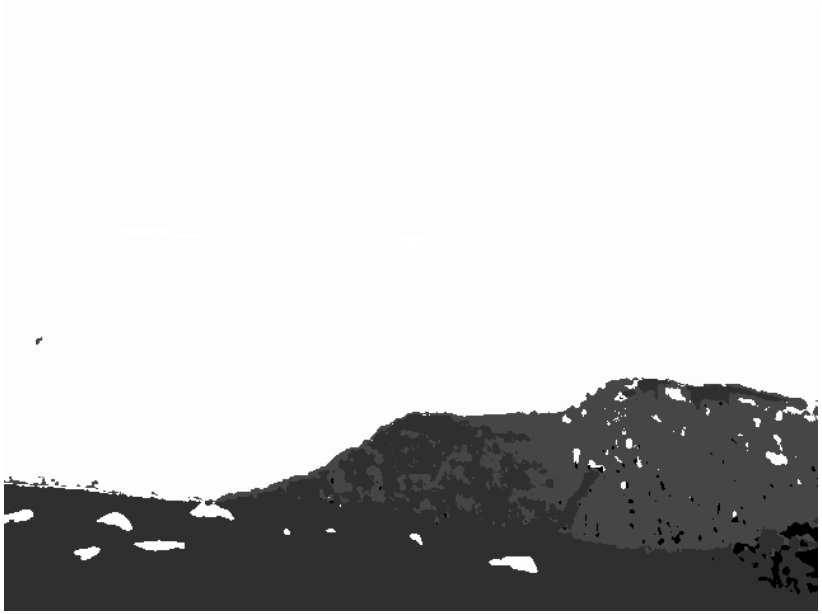
```
int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

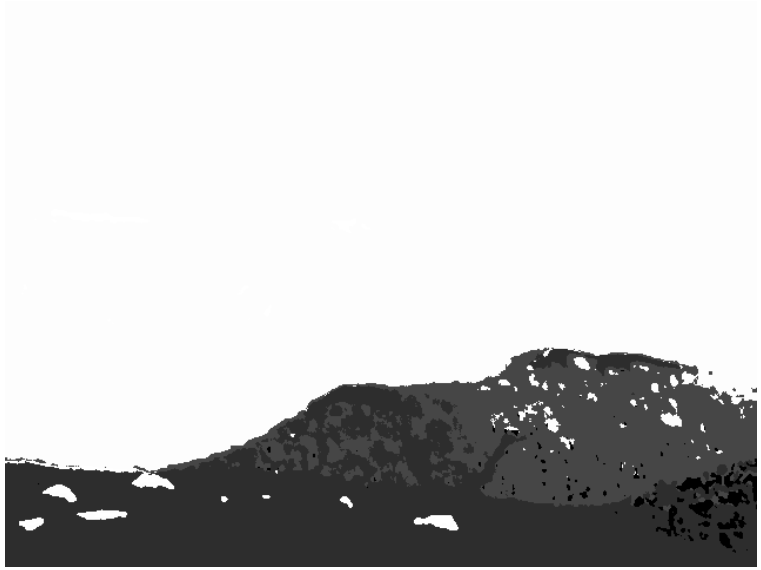
```
int kBands = 3; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
```

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

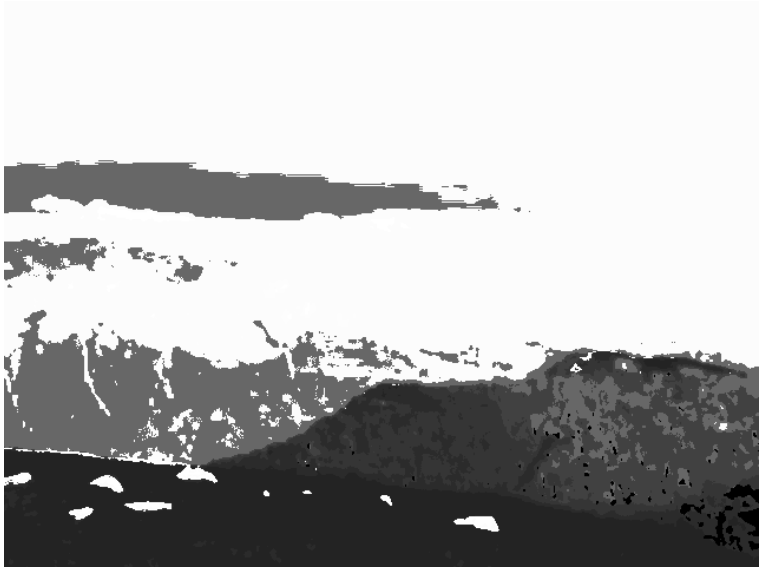
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

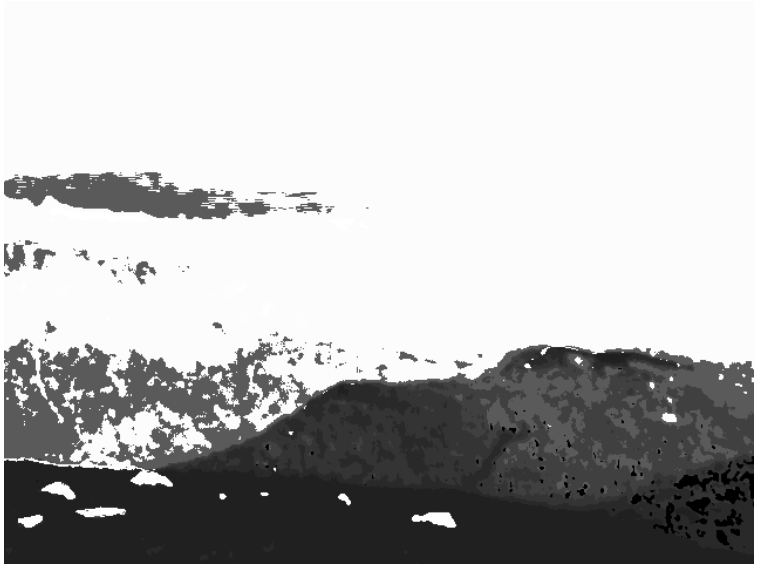
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);





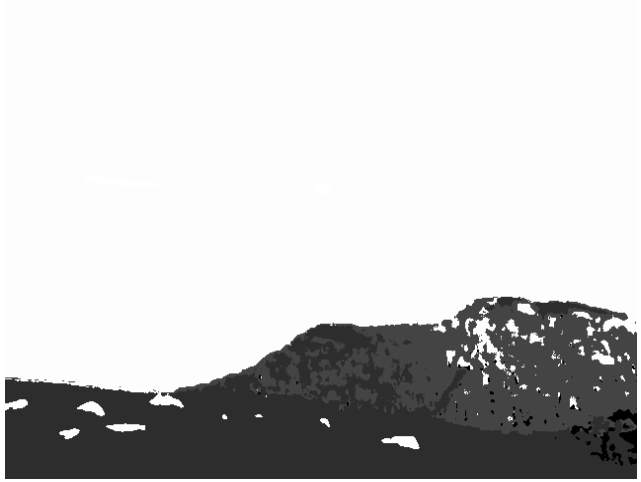
int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

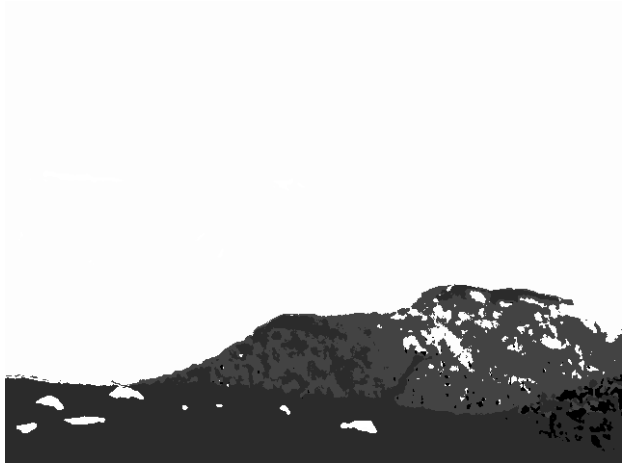




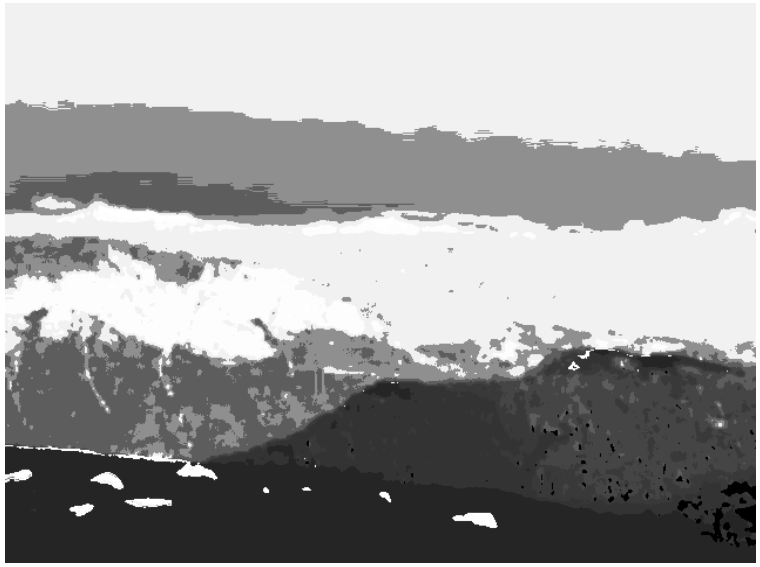
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

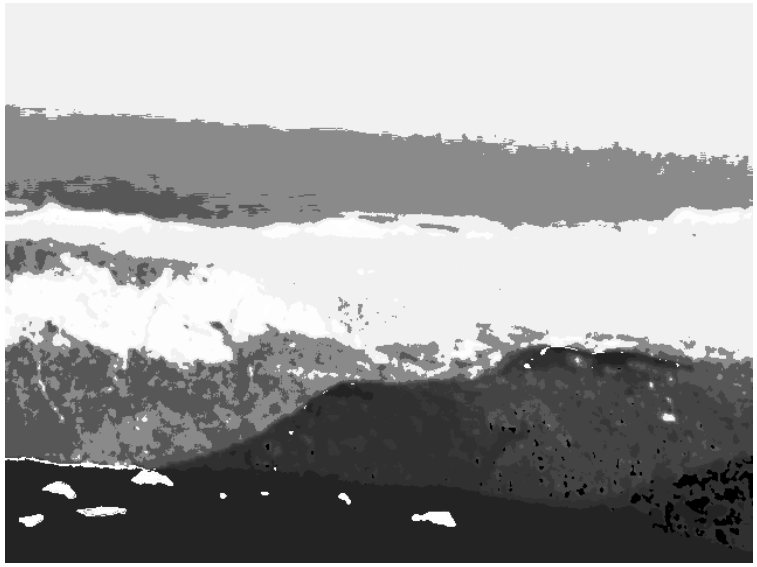
int kBands =3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);





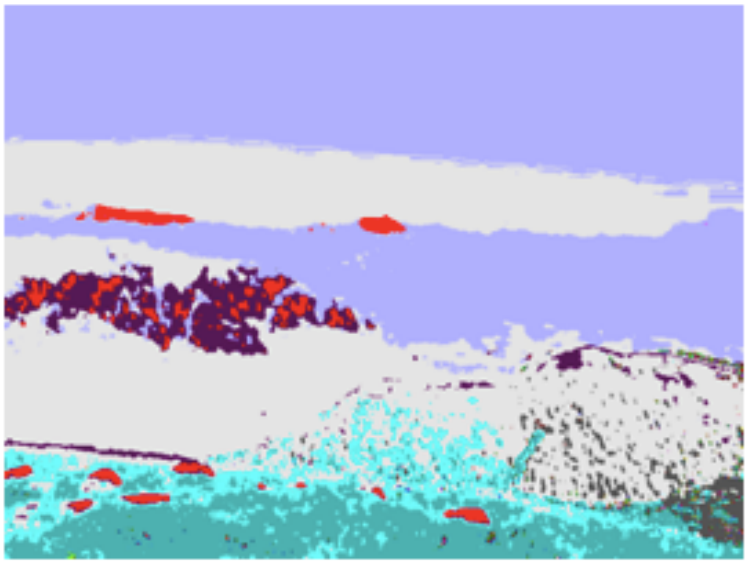
int kBands =8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq

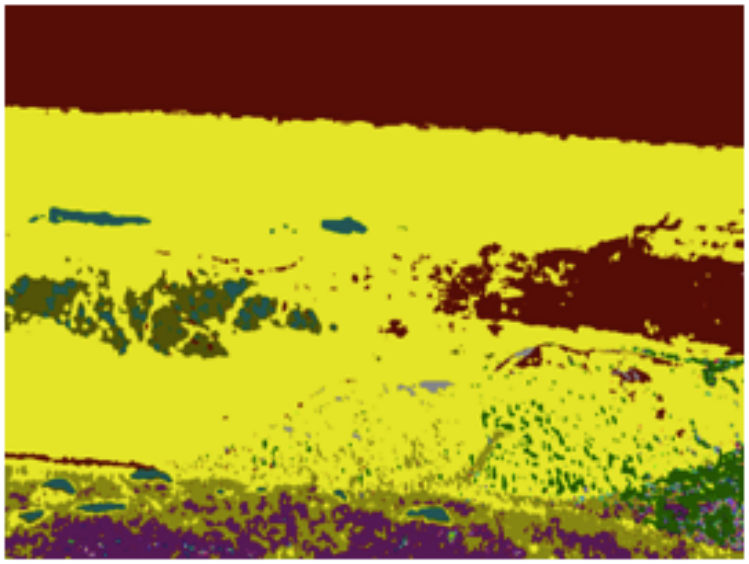




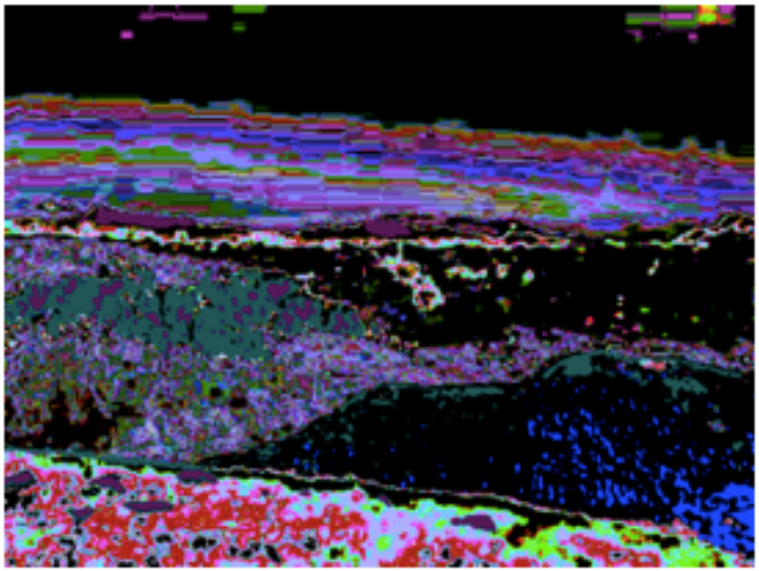
int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistogram(gsImg1, kBands);

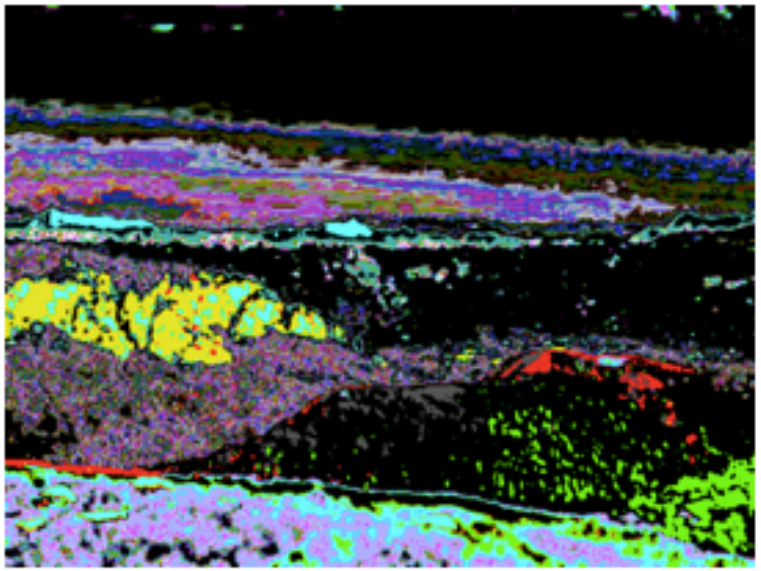
imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);



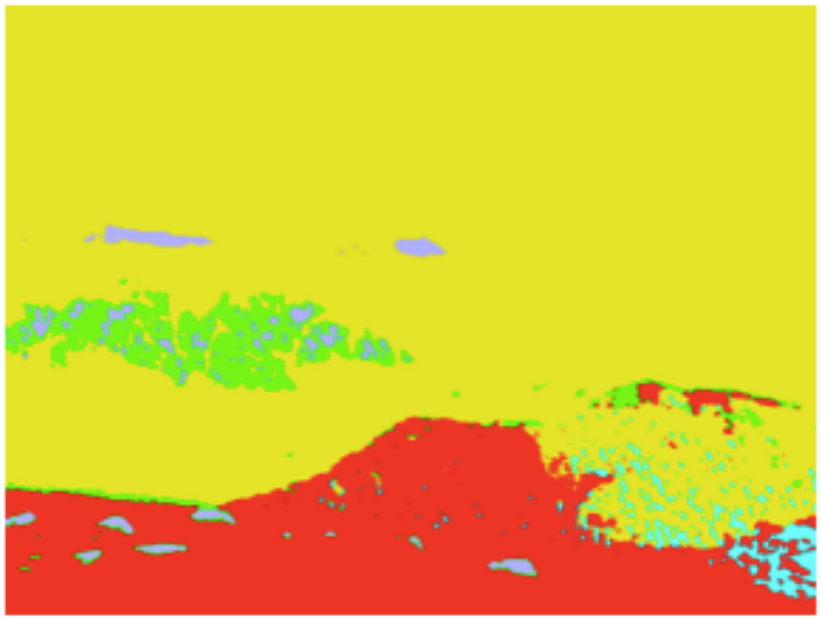


imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);

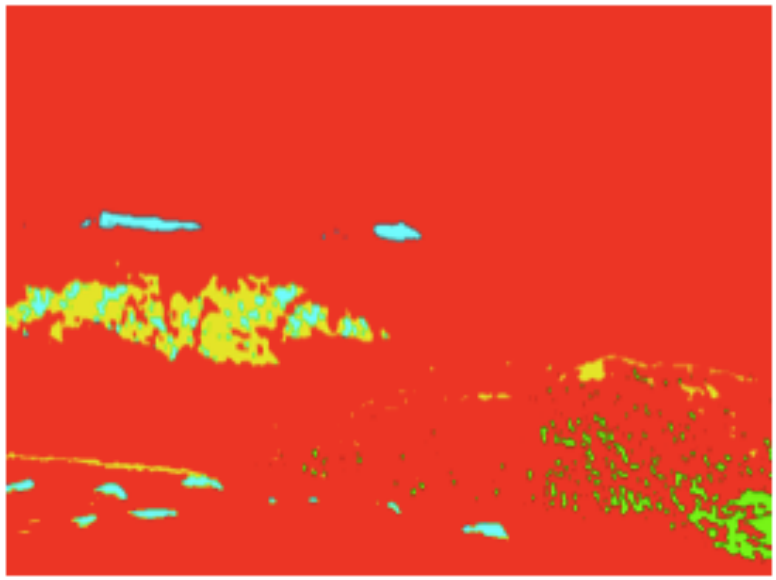


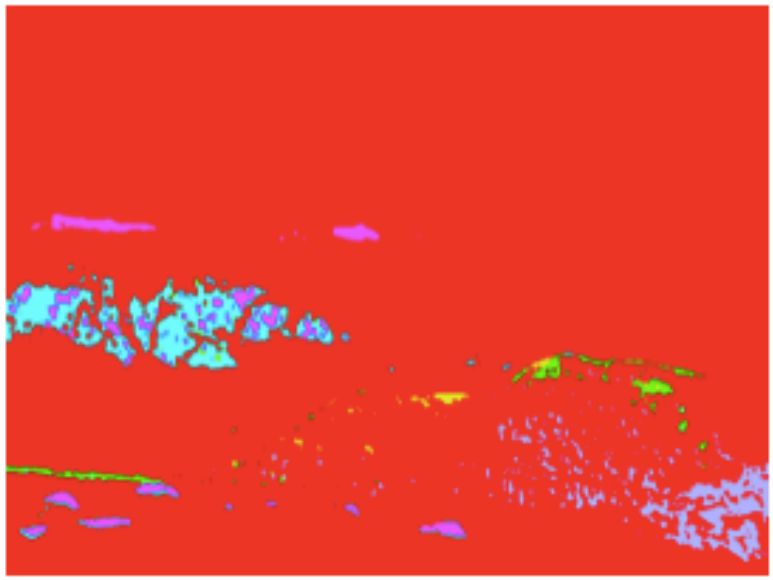


imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);



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

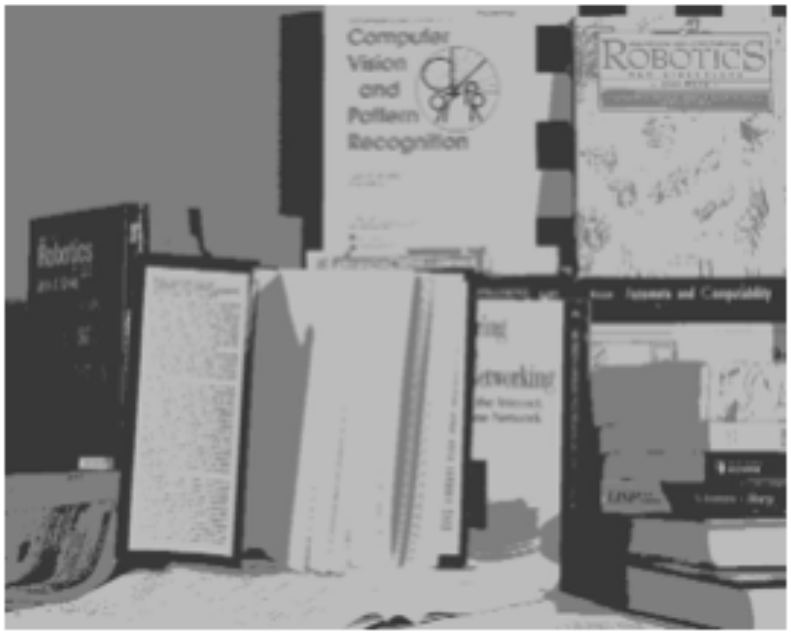


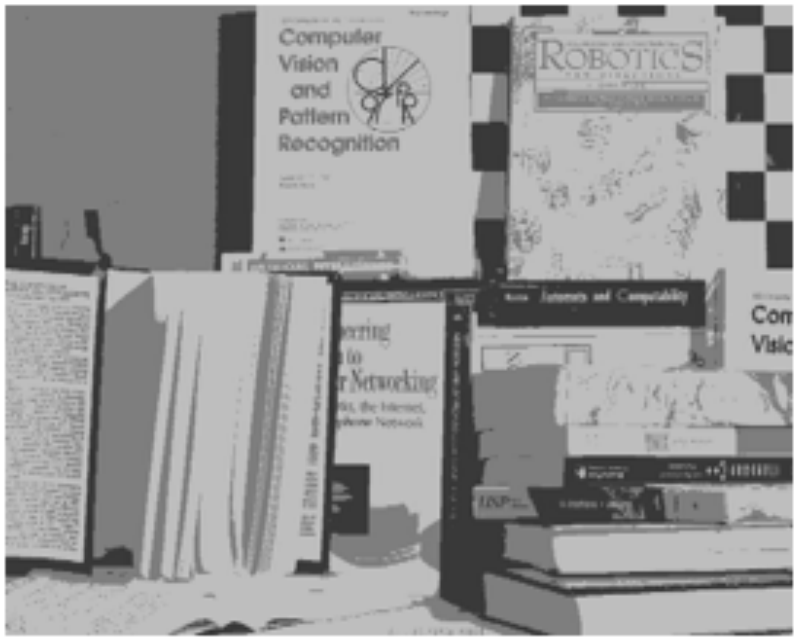


int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);



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



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

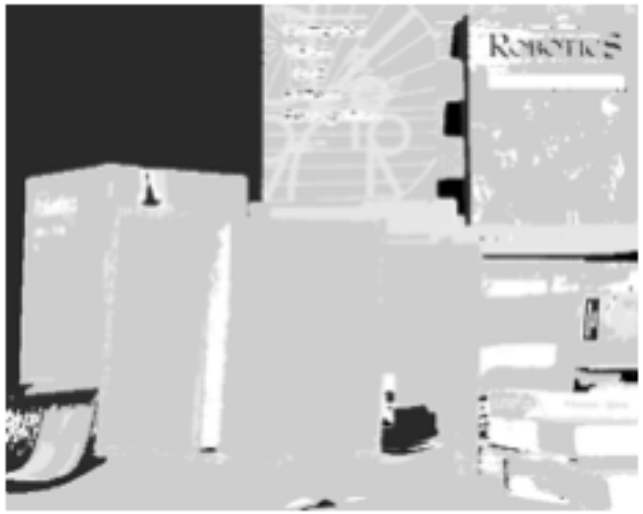


int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

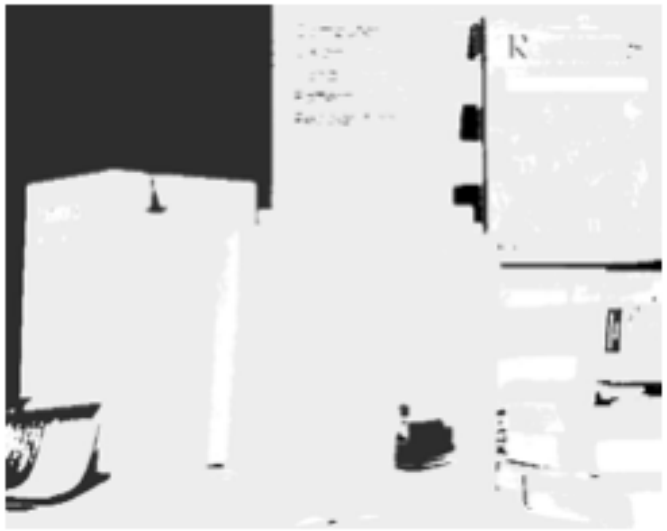
int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

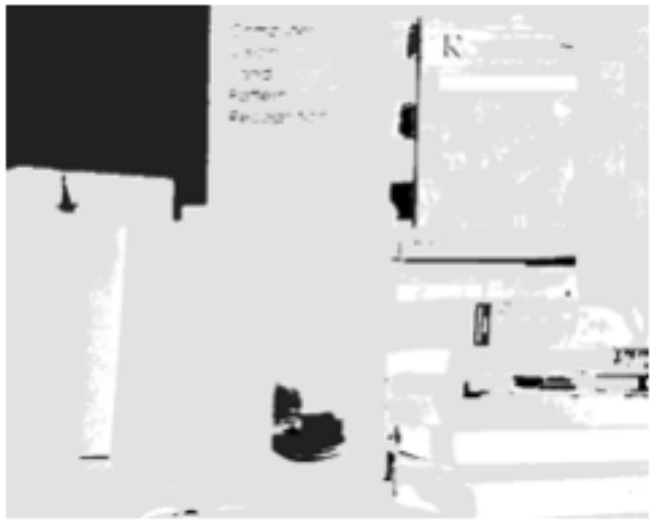


int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenHistEq(gsImg1, kBands);

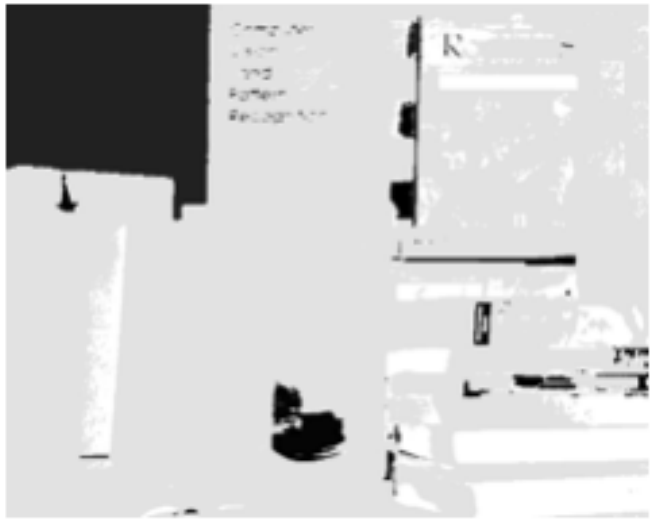



int kBands = 2; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

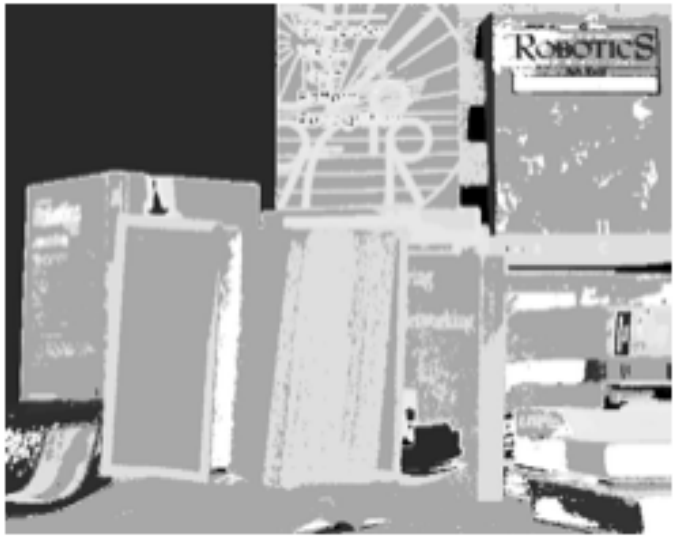


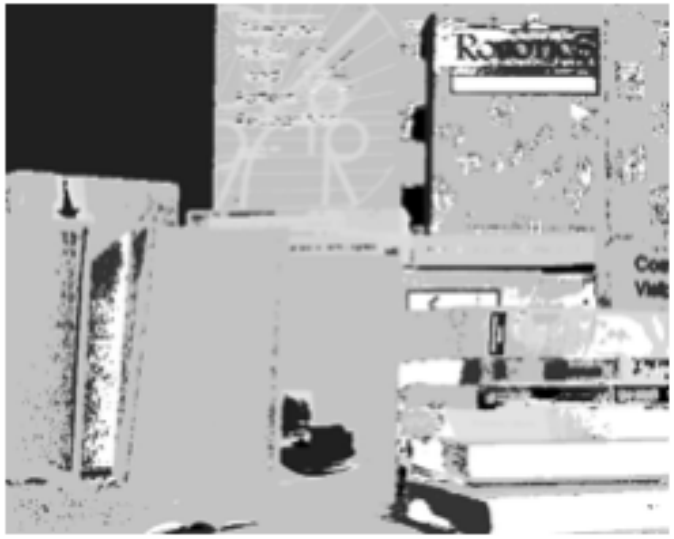


int kBands = 3; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

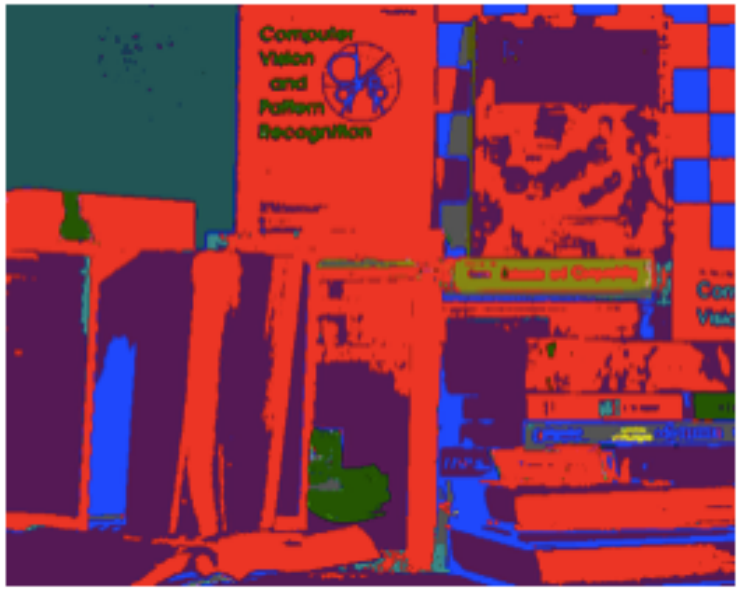


int kBands = 8; imageSegmentation.applyUsingCIEXYPolarThetaThenKMPPThenHistEq(gsImg1, kBands);

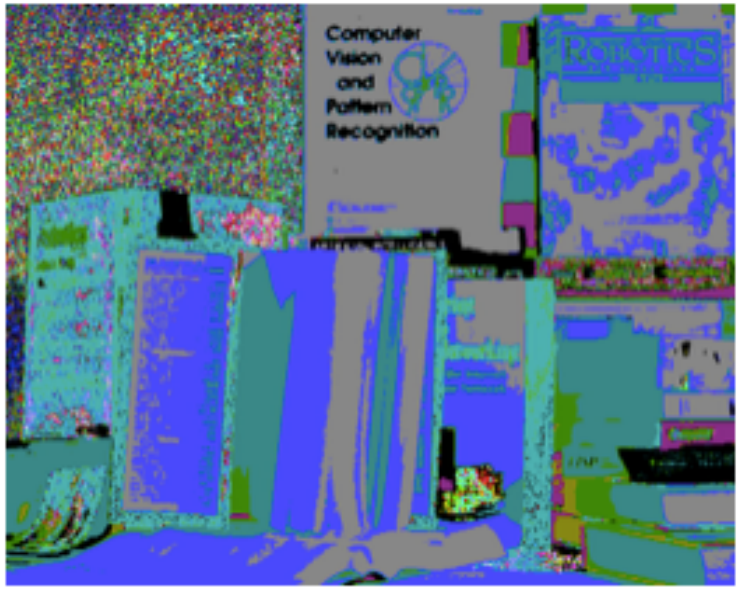


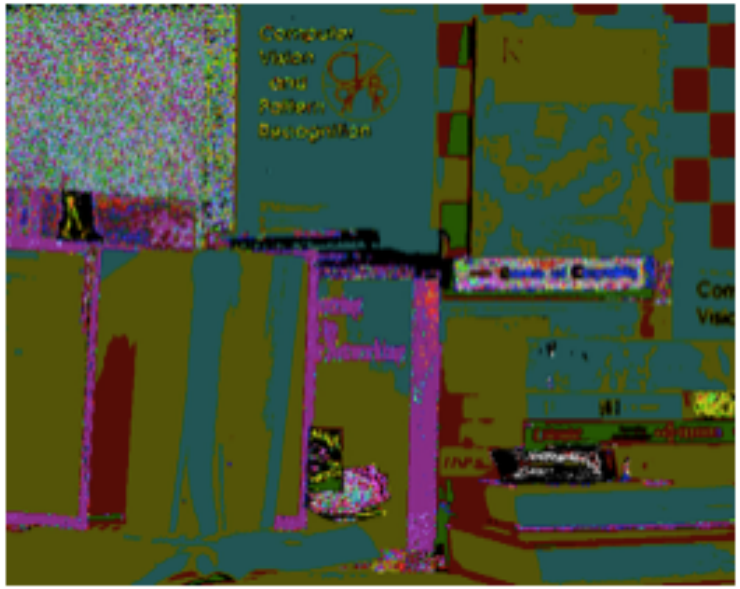



imageSegmentation.calculateUsingCIEXYAndClustering(gsImg1, true);

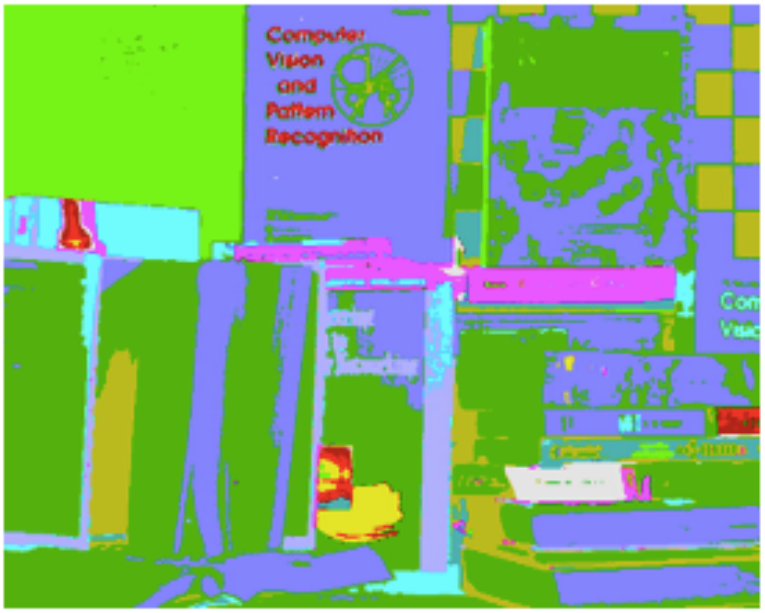




imageSegmentation.calculateUsingPolarCIEXYAndClustering(gsImg1, true);

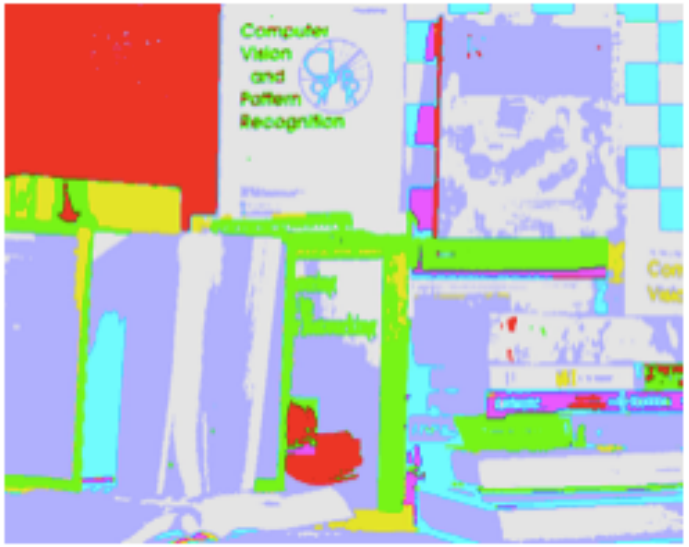


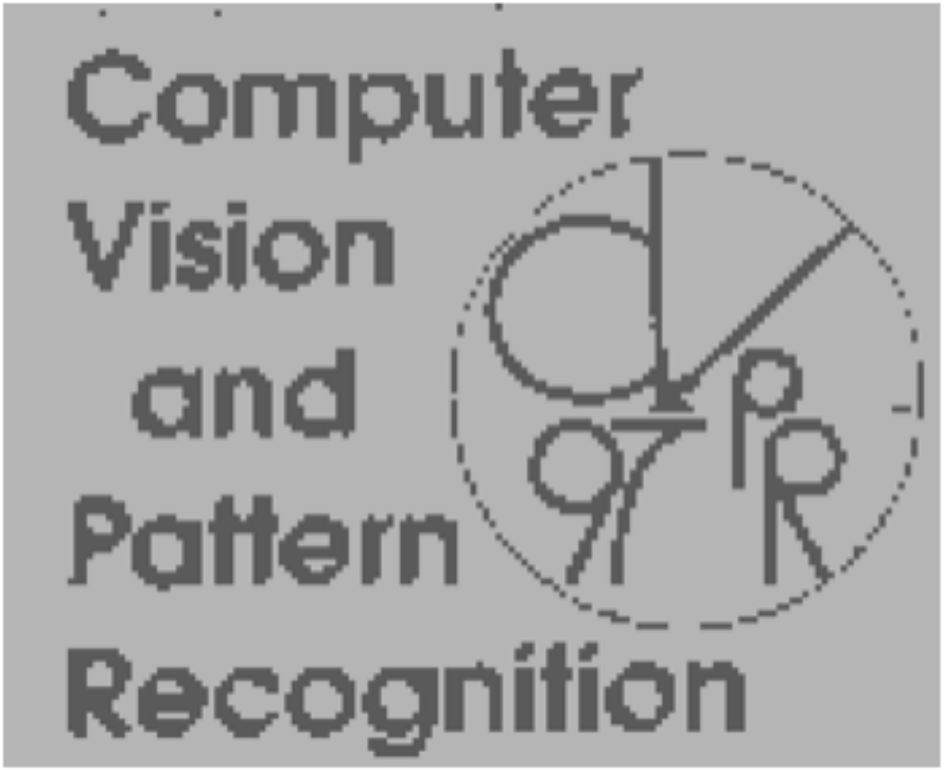
imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, true);

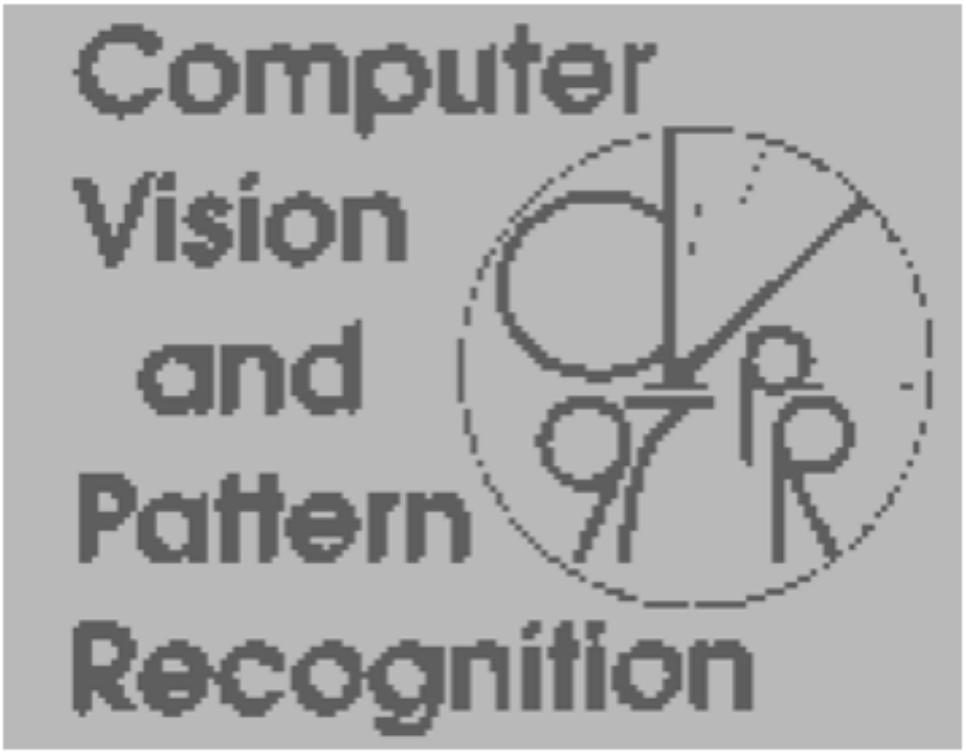




imageSegmentation.calculateUsingPolarCIEXYAndFrequency(gsImg1, 0.2f, true);

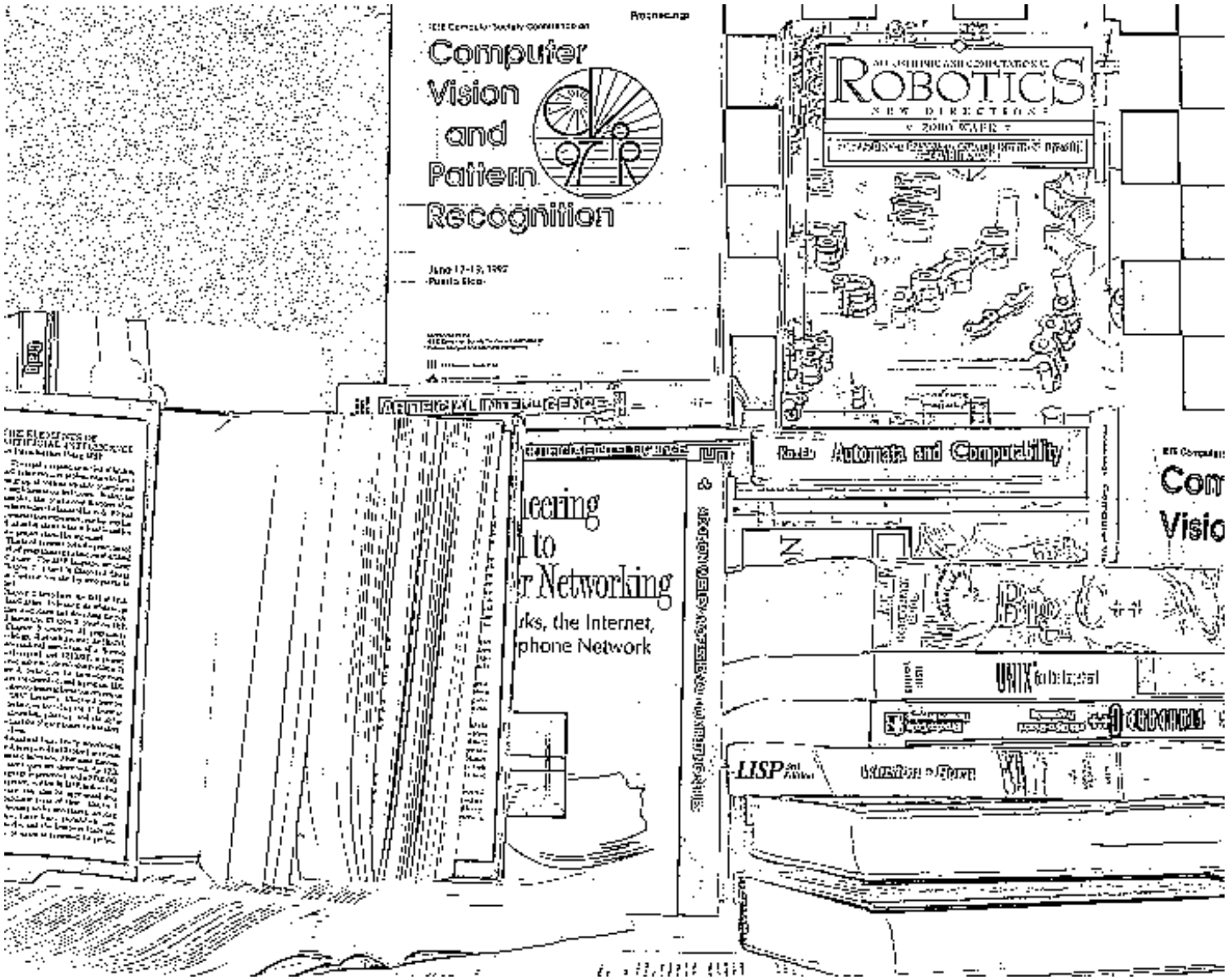



int kBands = 2; imageSegmentation.applyUsingKMPP(gsImg1, kBands);
zoom in

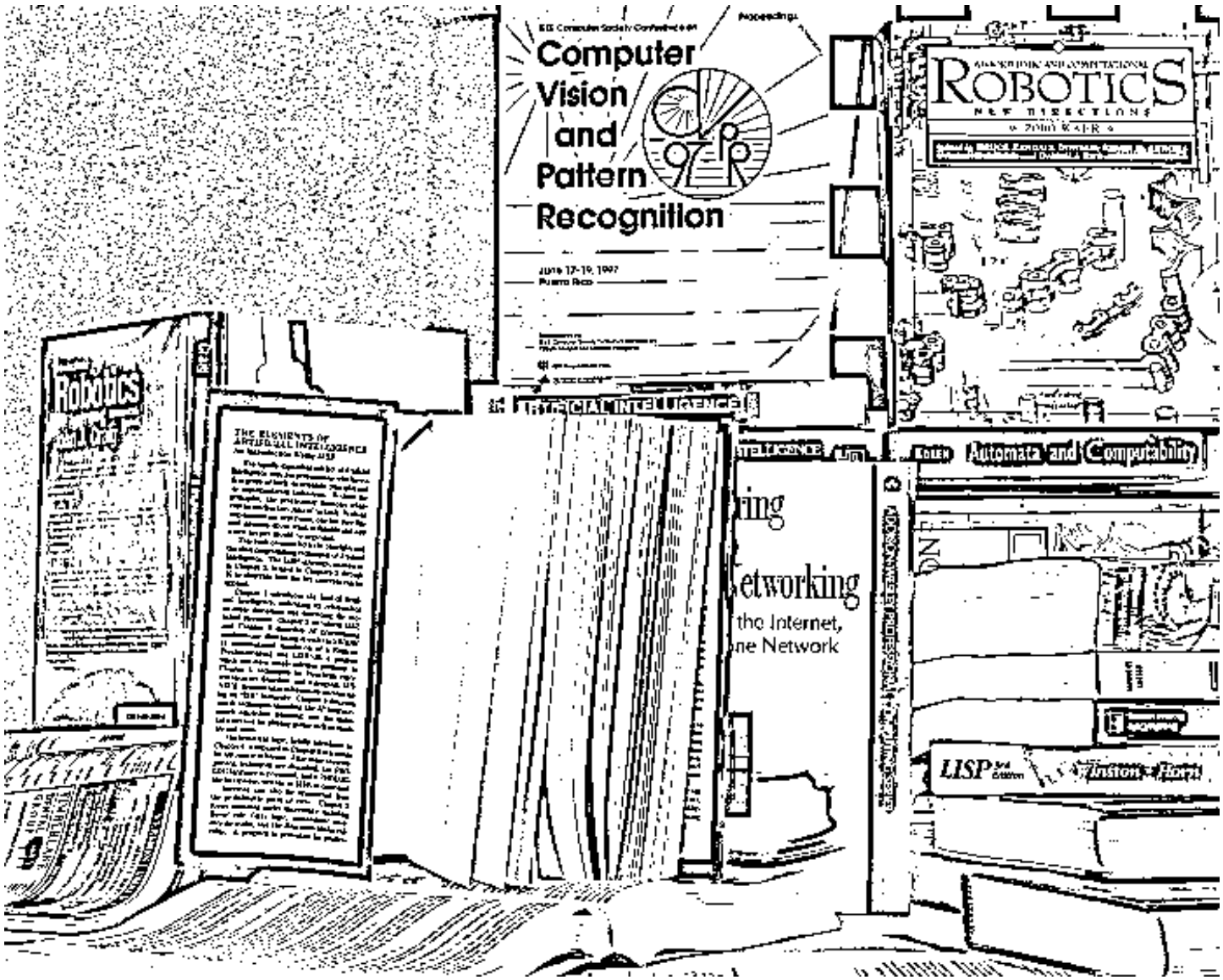


# adaptive means, h=1

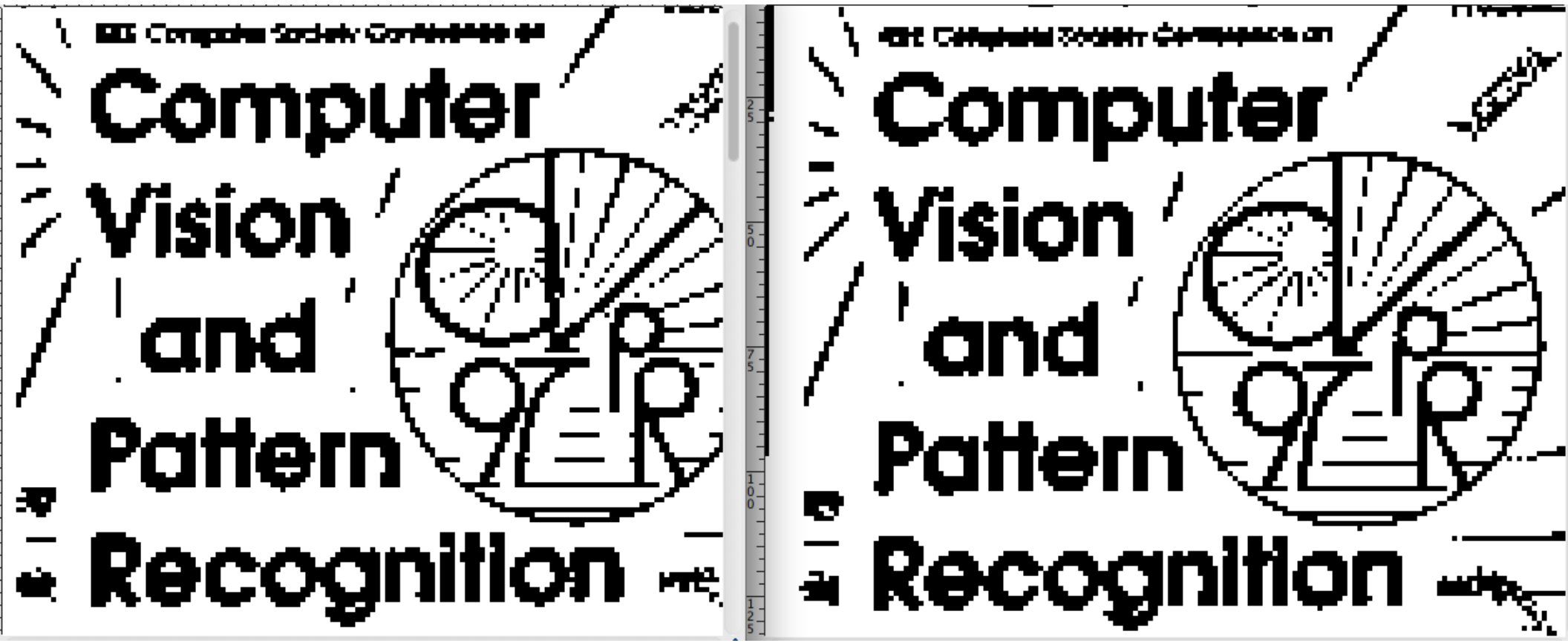


# adaptive means, h=3

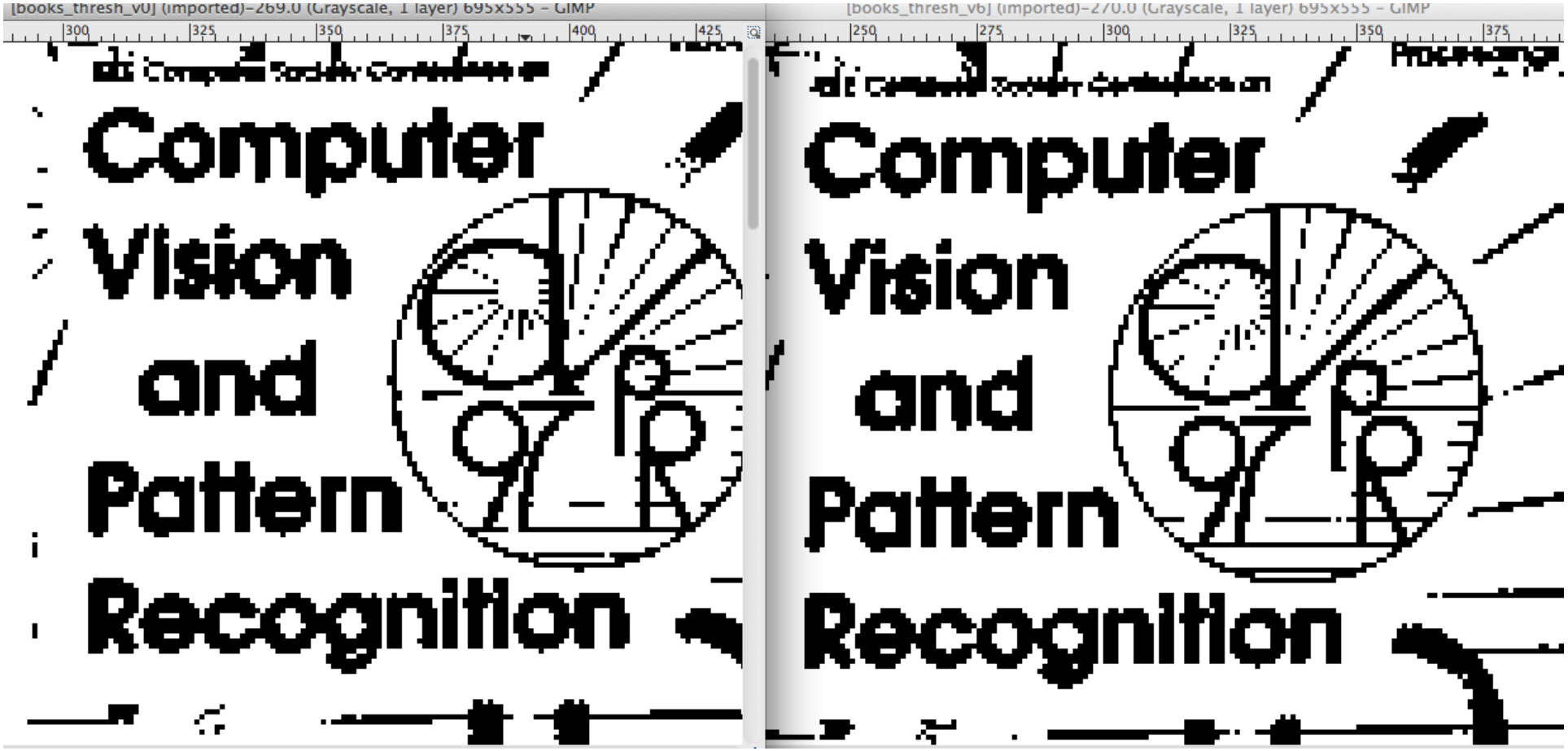


# adaptive means, h=5

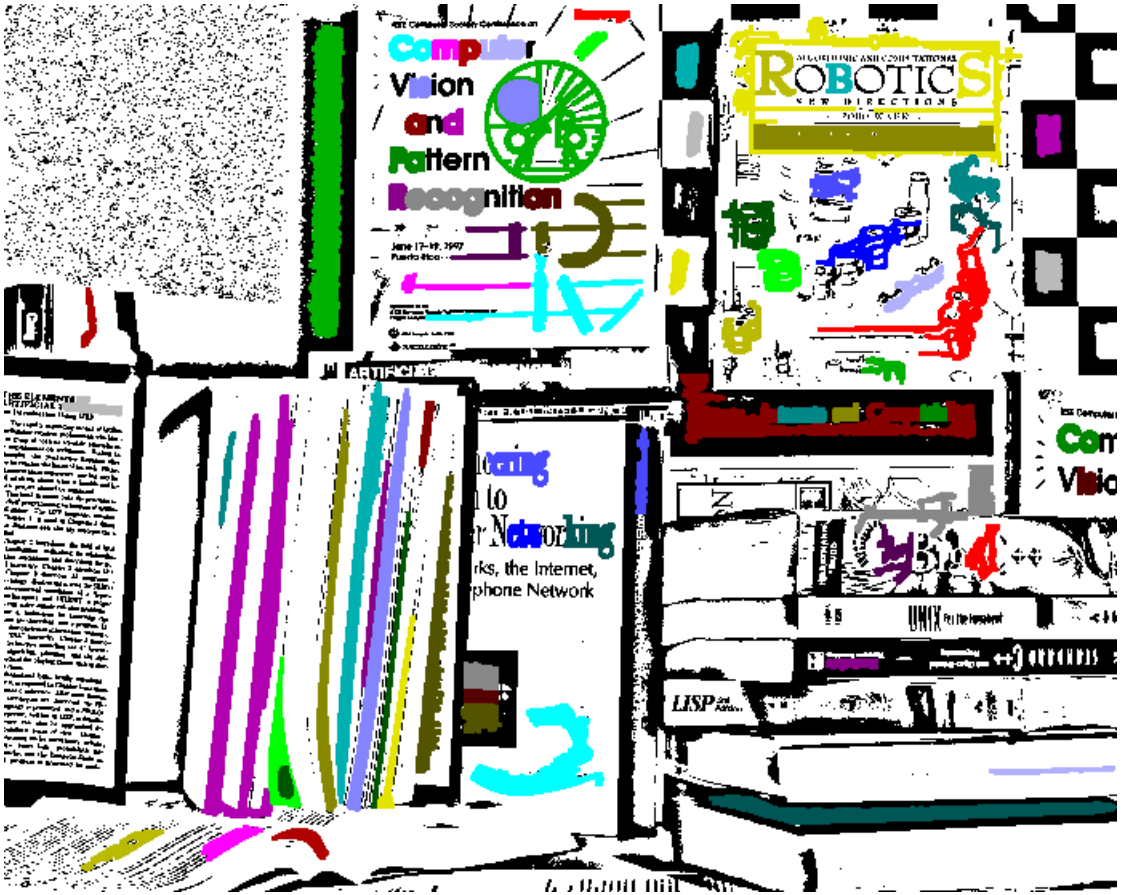
adaptive means, h=7



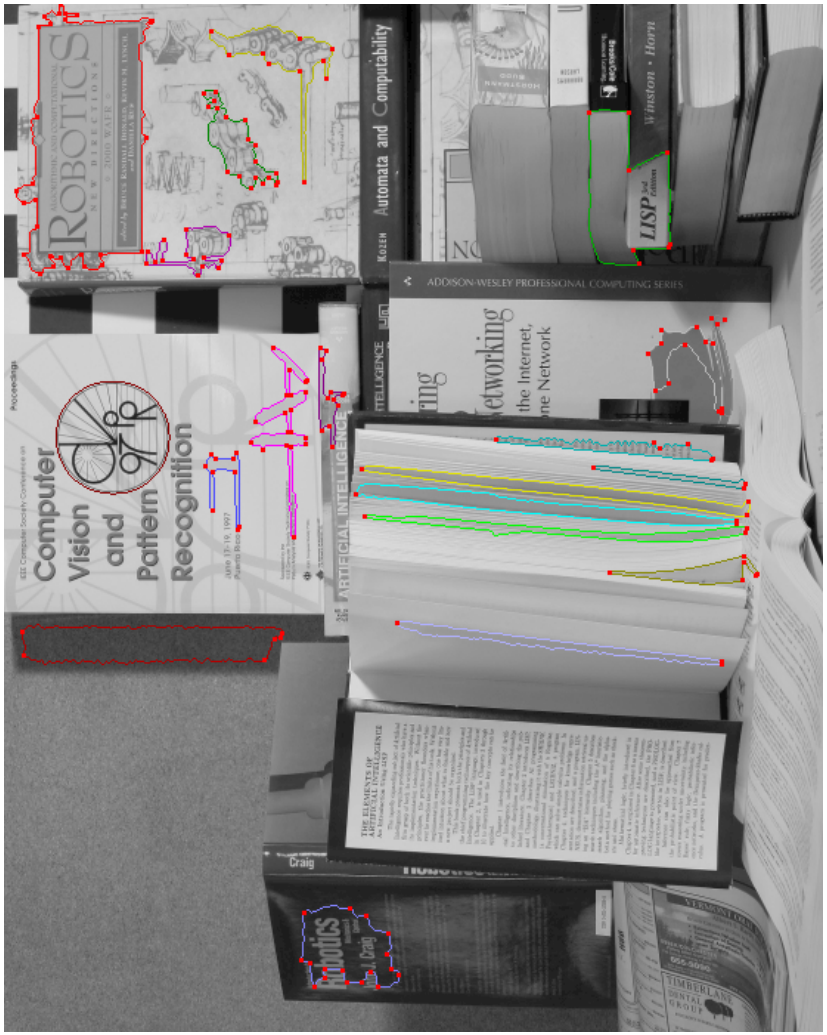
# adaptive means, h=11 for adaptive to join close letters

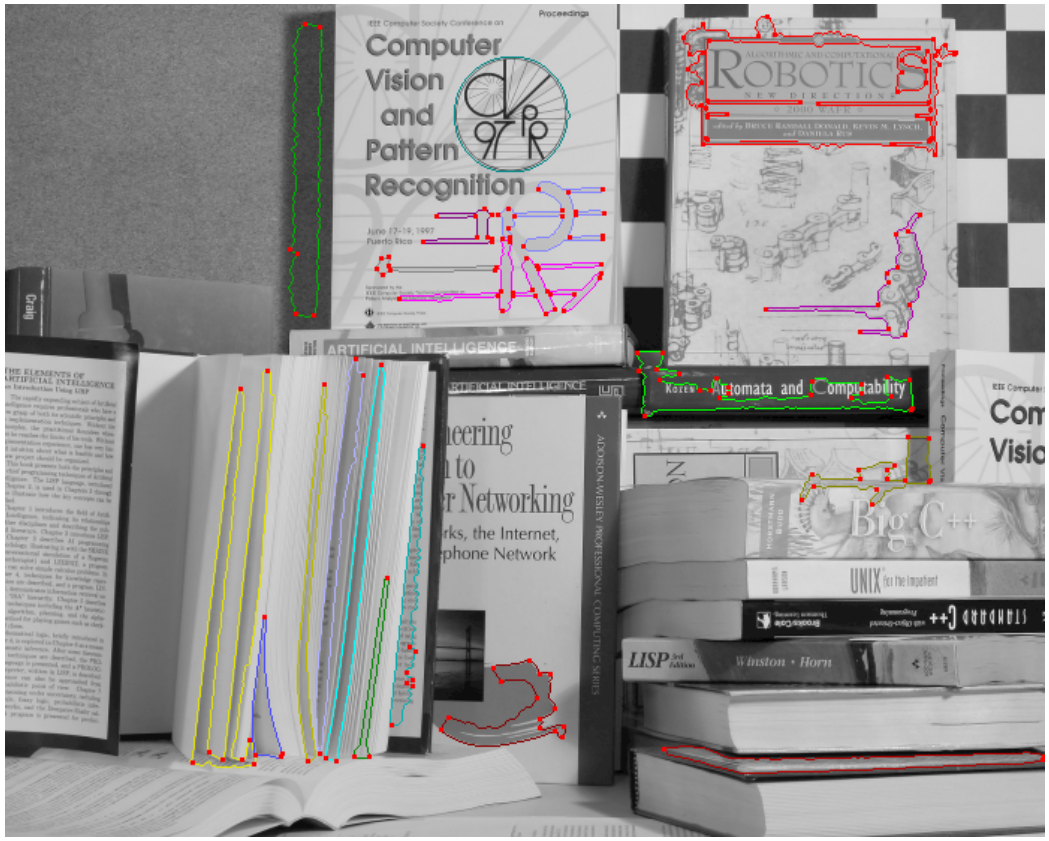


# adaptive means, h=11 blobs (extr using scale calc code)

# adaptive means, h=11 blob corners

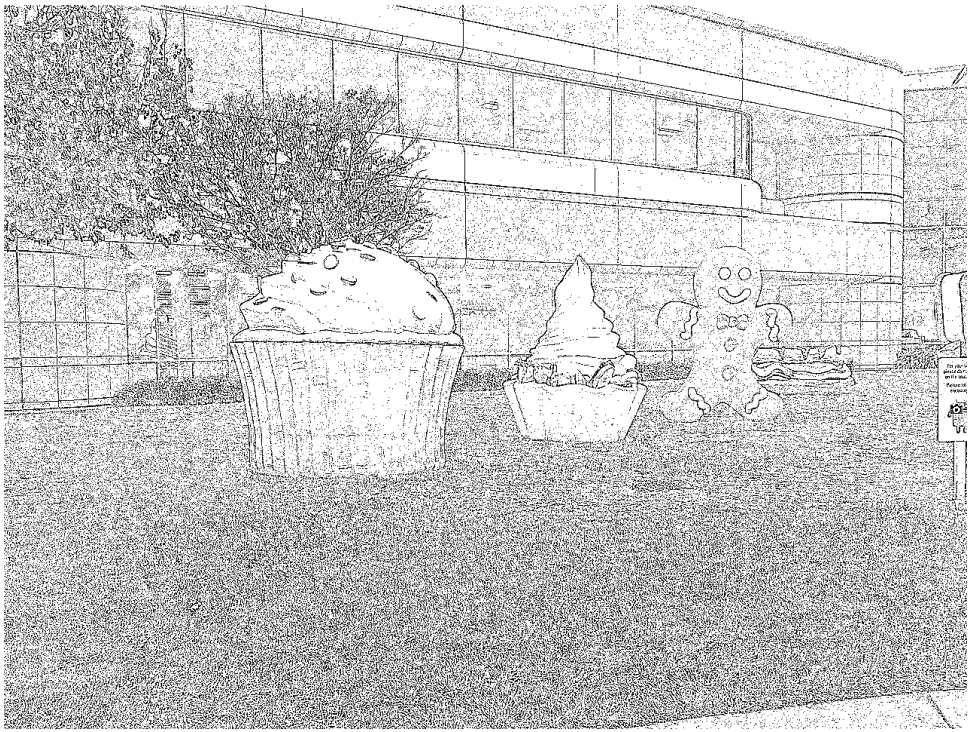


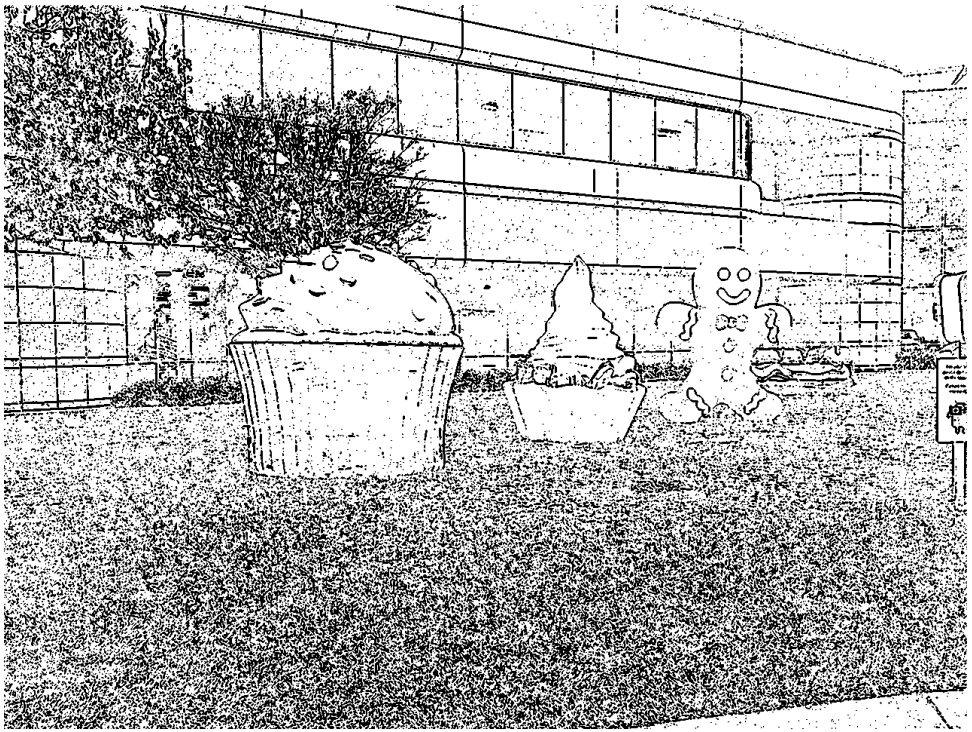

the letters are filtered out due to large nPerimeter compared to nInternal

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

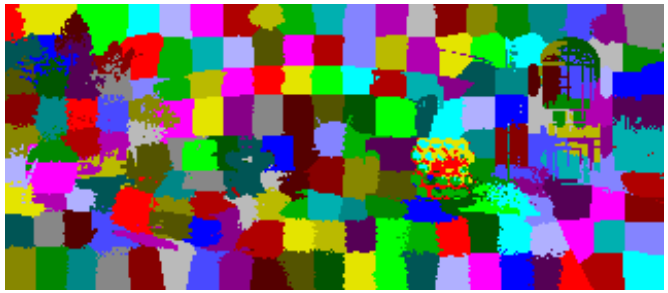
adaptive means, h=1

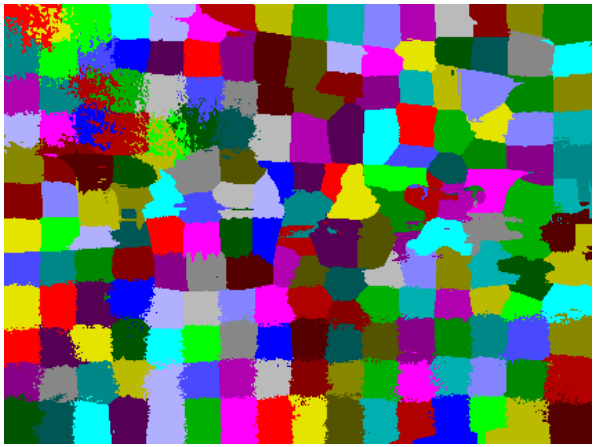
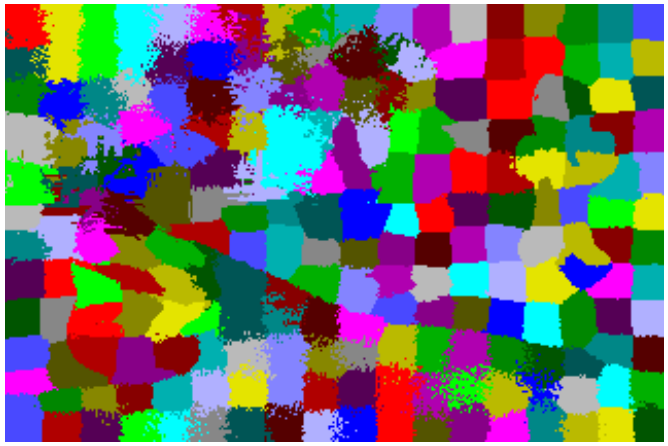


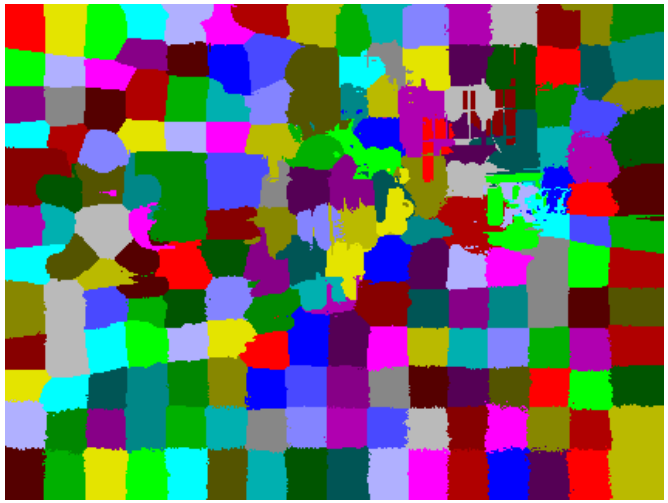
two median smooth with h=2



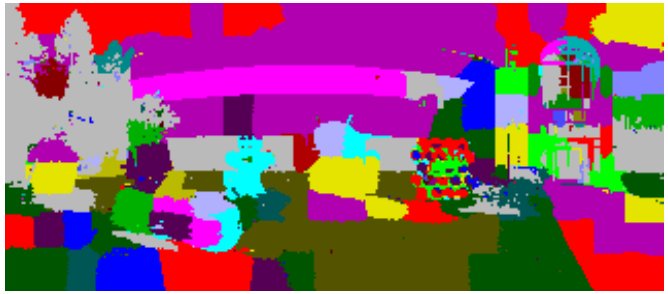
# SLIC super pixels algorithm

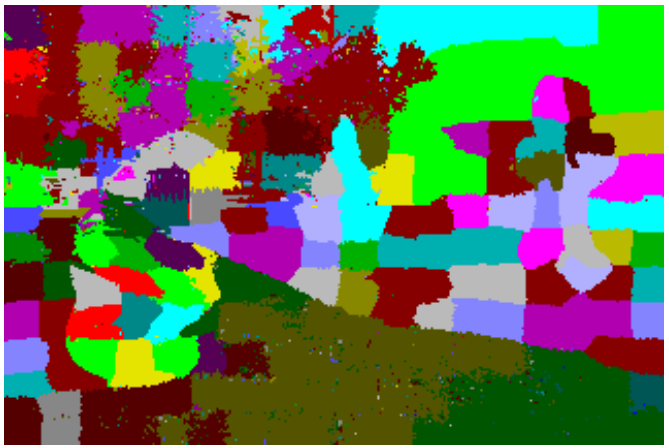


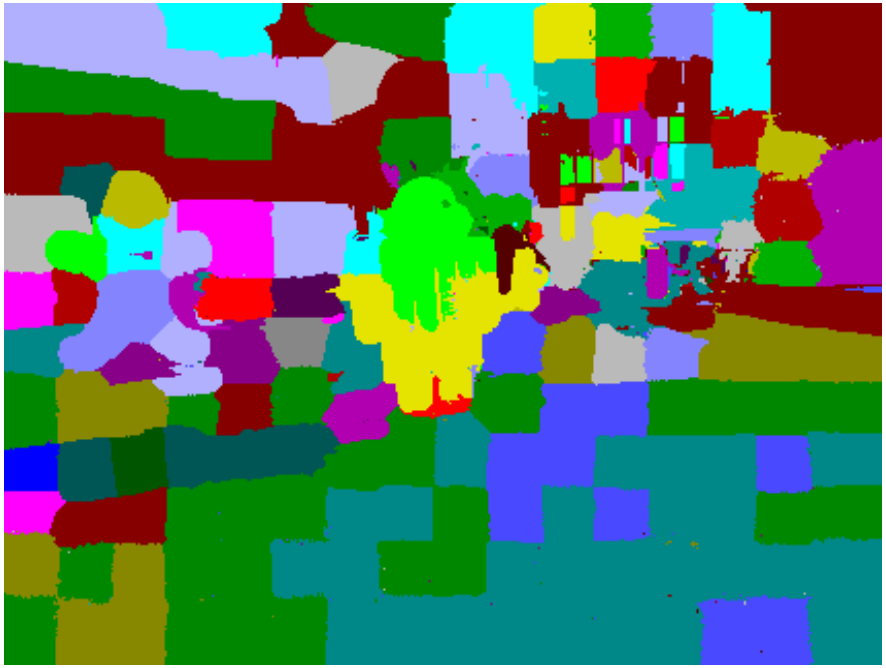



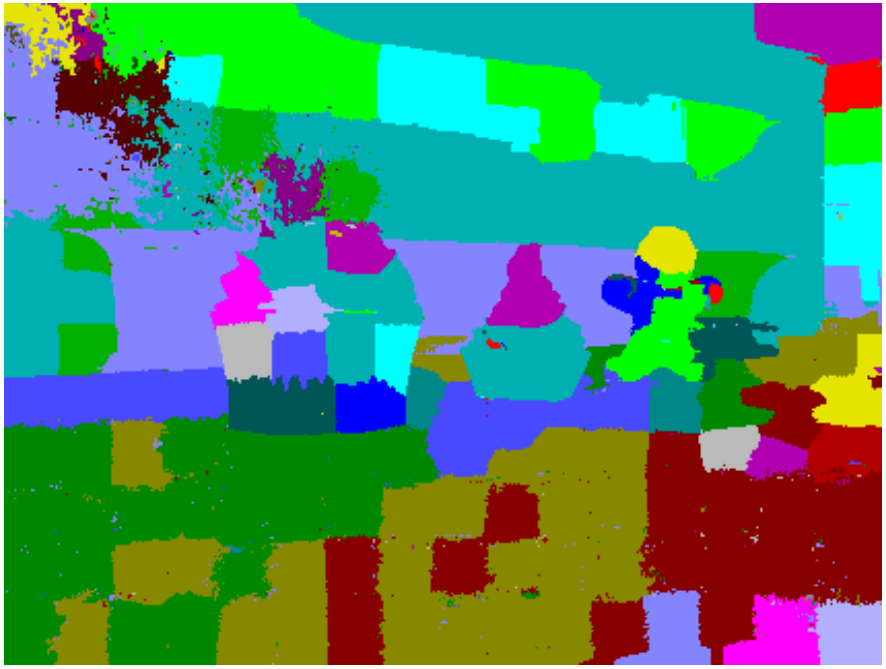




# CIE XY Theta Clustering of the super pixels

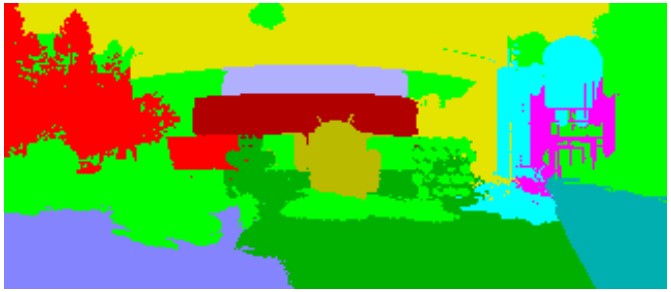


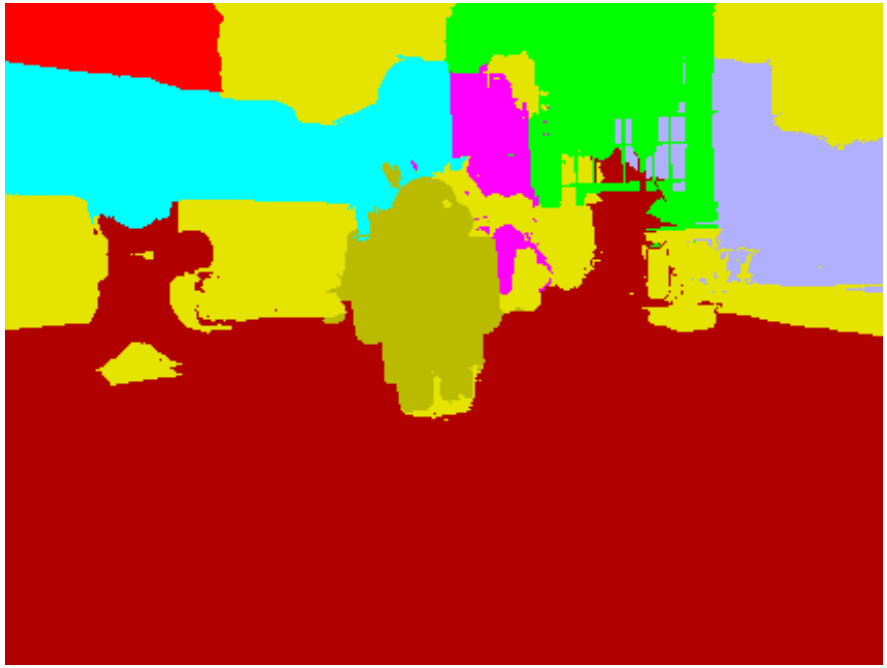


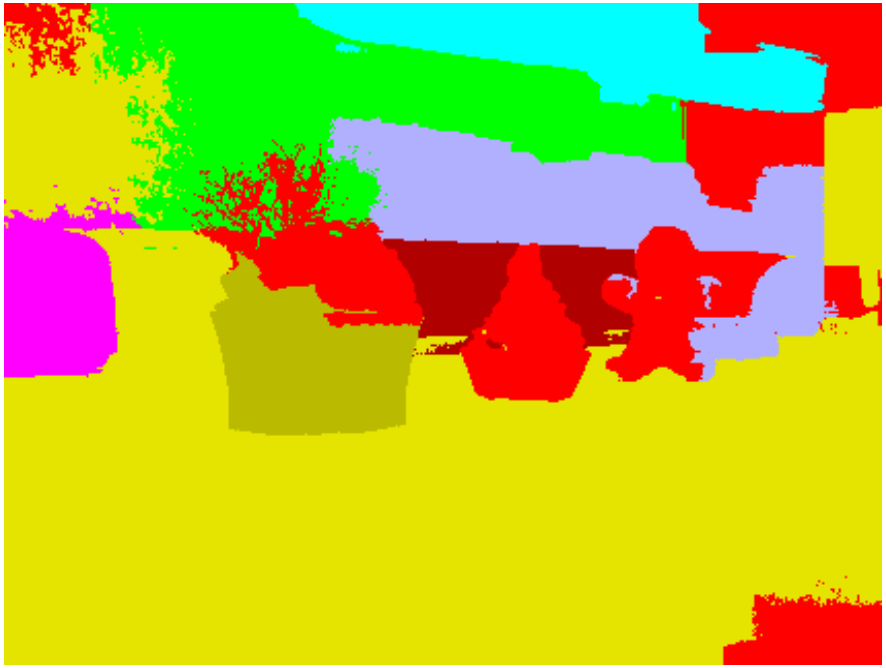




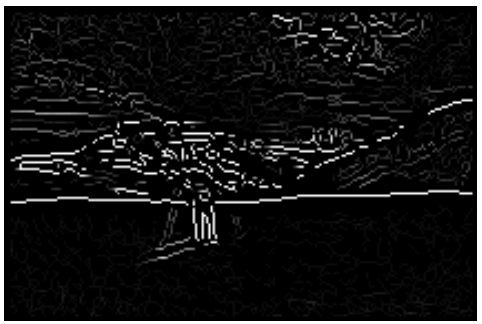
# Normalized Cuts w/ HSV cs on the super pixels



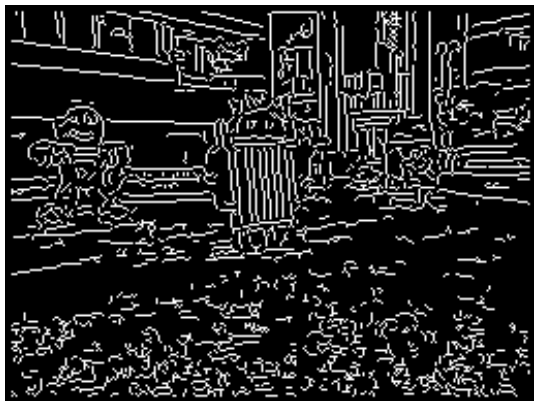


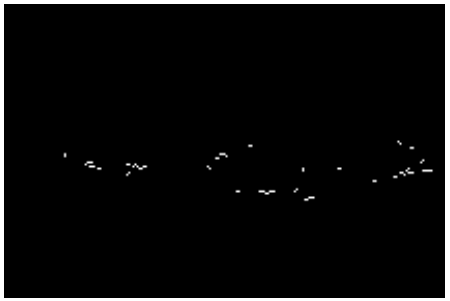


# canny greyscale

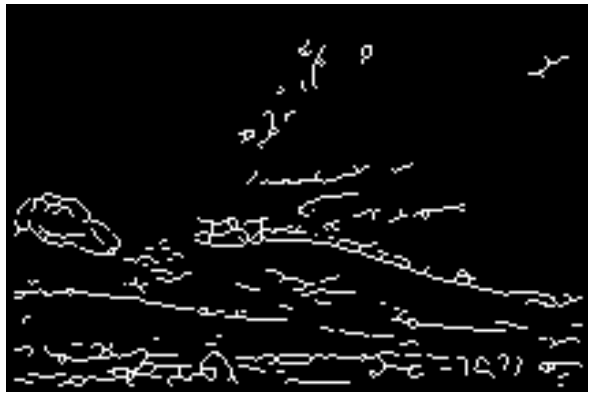


# canny deltaE 2000





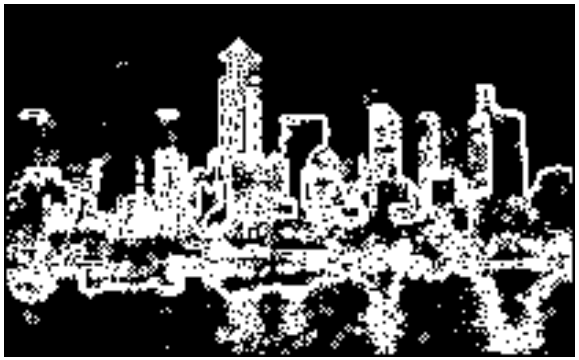
# phase congruence

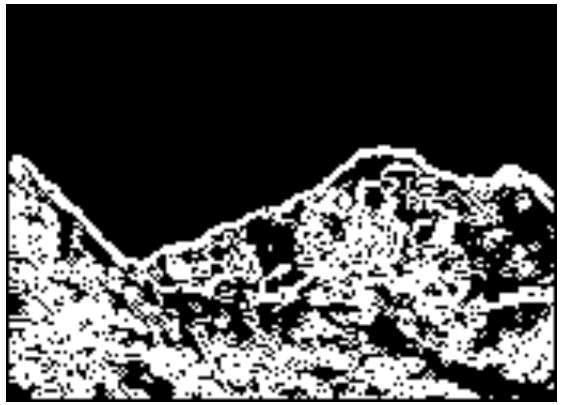


# binary sobel “H” of LCH with 20 degree threshold

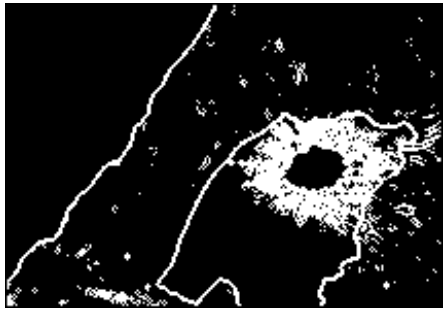
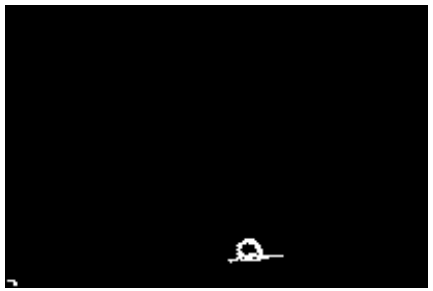


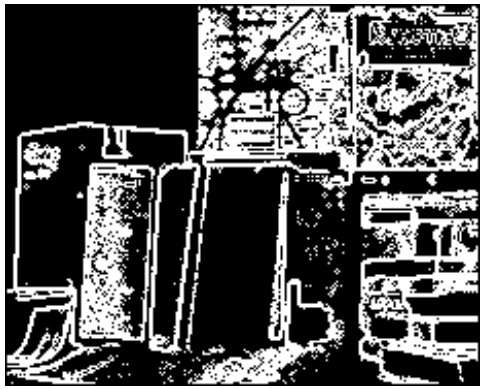




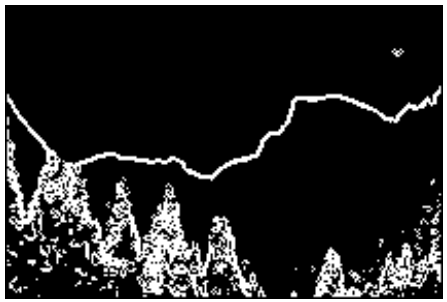



# binary sobel “H” of LCH with 20 degree threshold





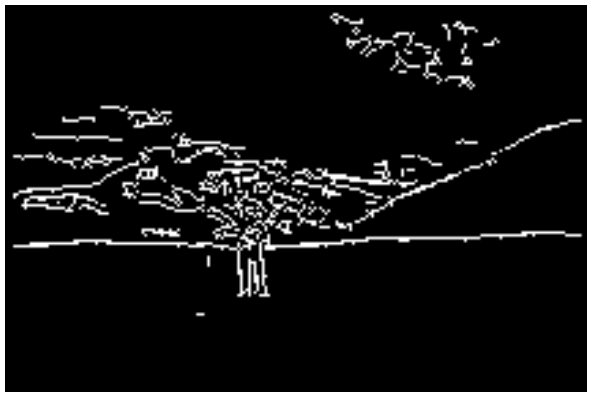


# Canny edges of “L” and “C” from LCH colorspace



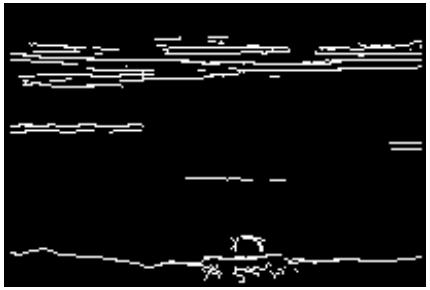




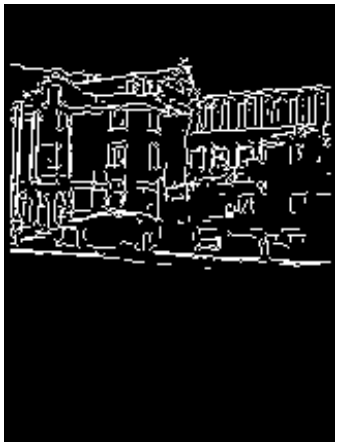
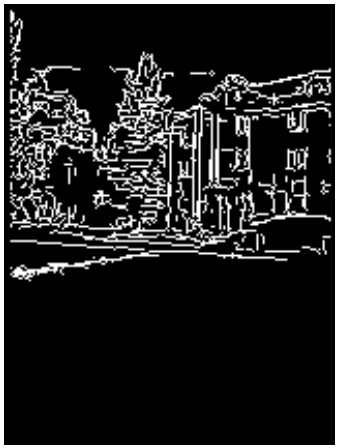


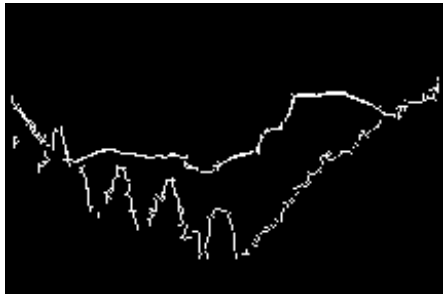

# Canny edges of “L” and “C” from LCH colorspace

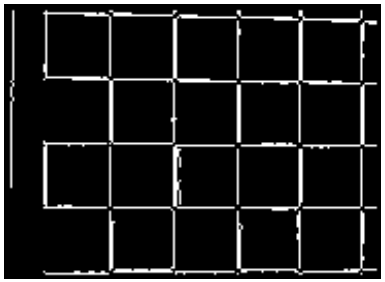




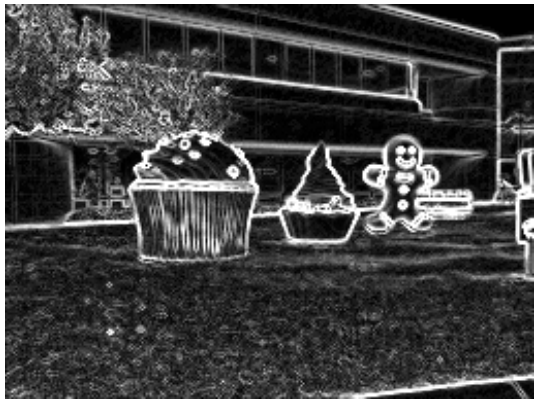


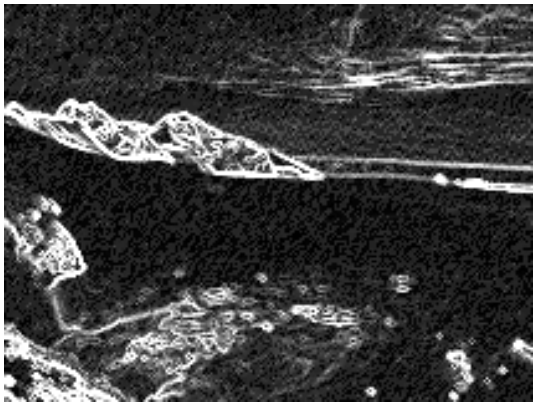



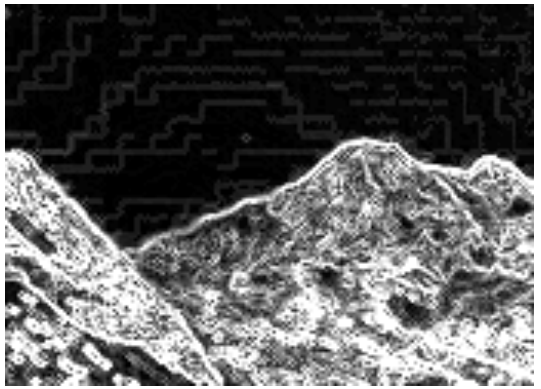




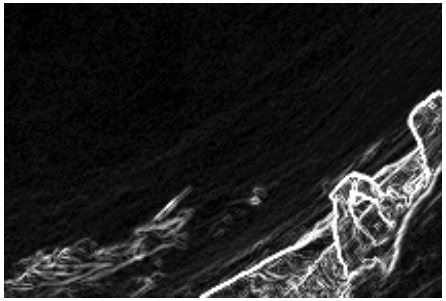
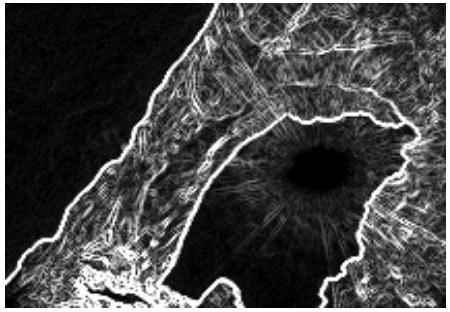
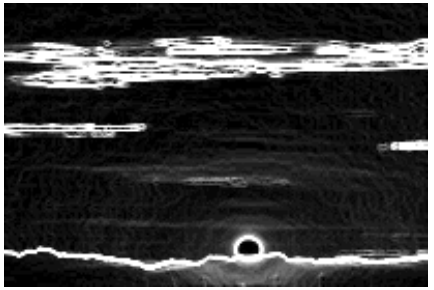
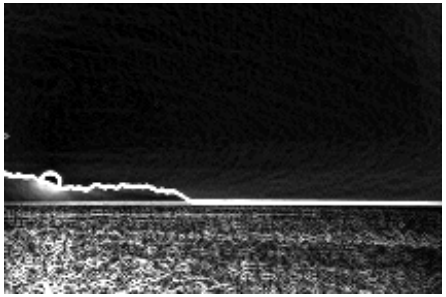
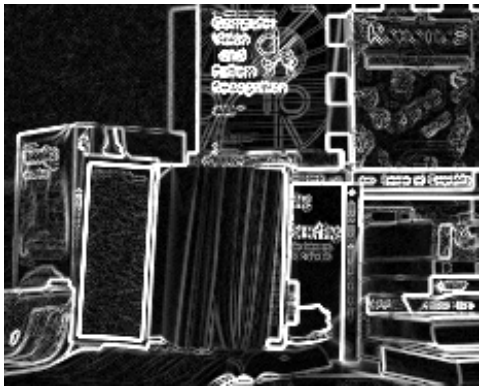
# Sobel edges of “L” + “C” from LCH colorspace (stretched by x 4)

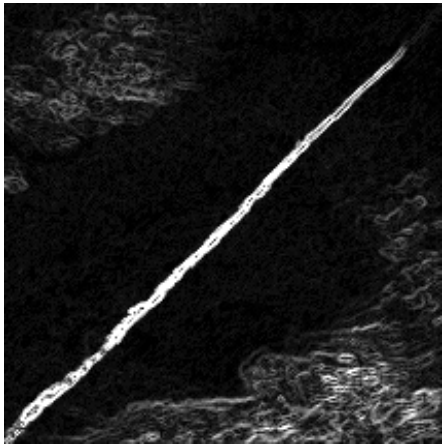
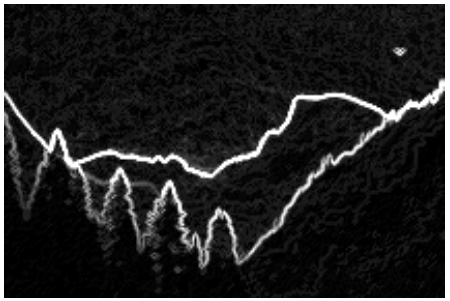
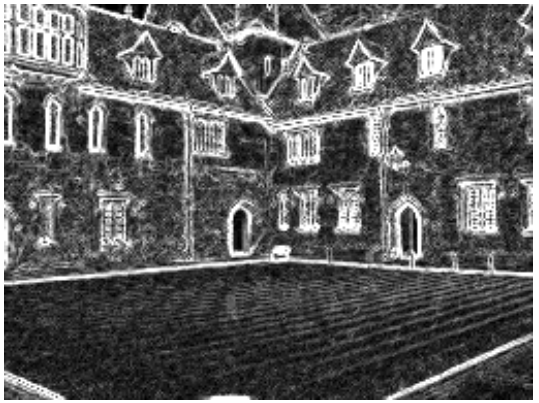
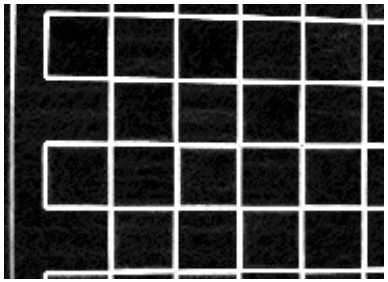




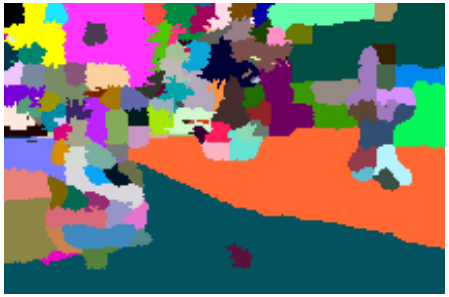


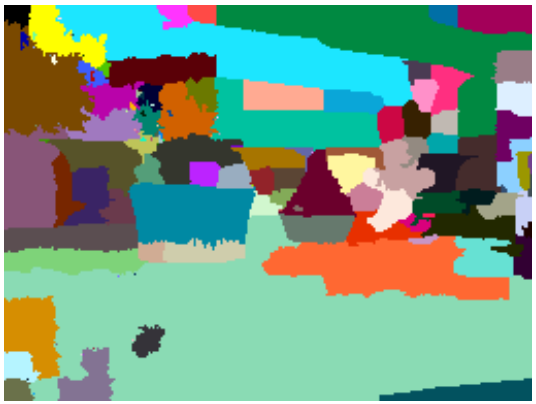
# Sobel edges of “L” + “C” from LCH colorspace (stretched by x 4)

# super pixels and moderate merging by HSV color

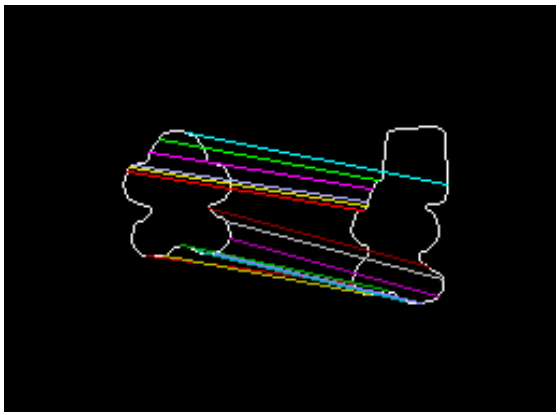



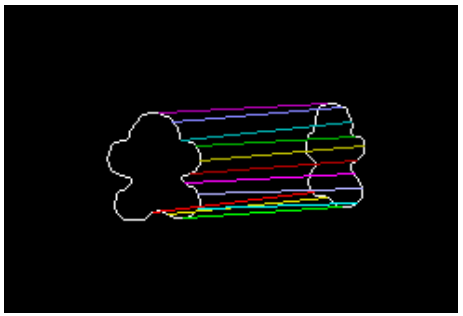


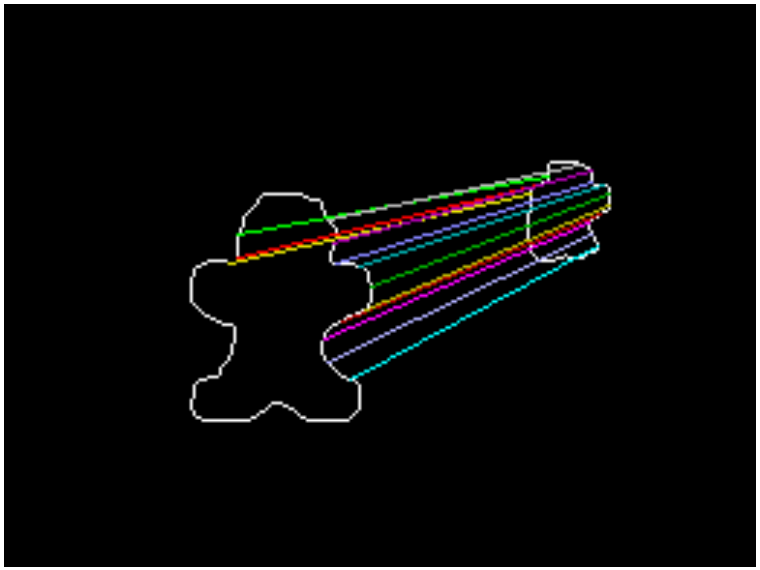


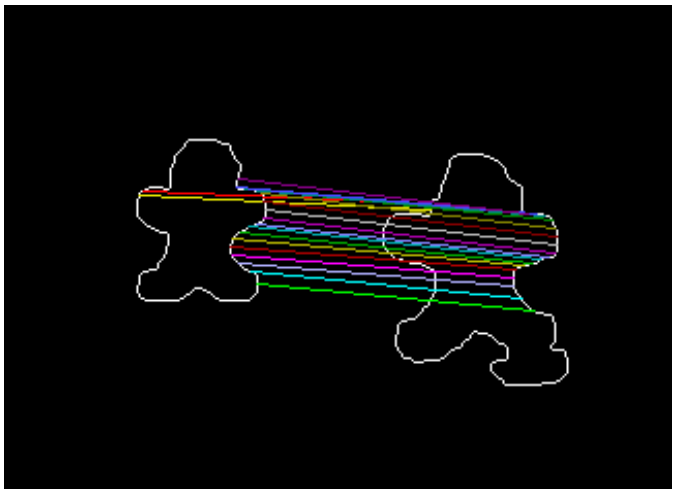
# Partial Shape Matching of the gingerbread man

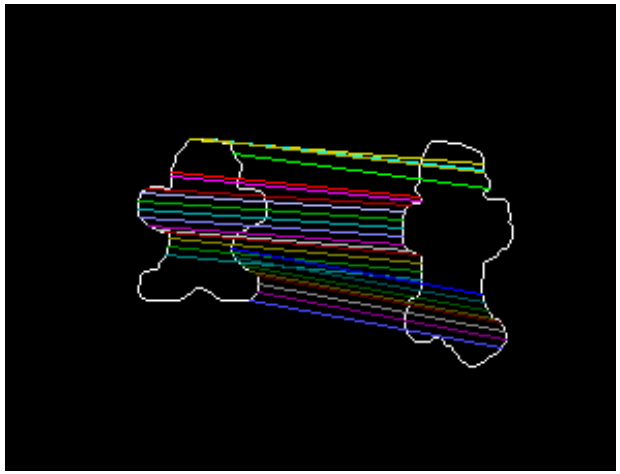
extract ordered borders of shapes, blur them by one or two sigma, then ordered bipartite matching algorithm using difference of chords as a descriptor.

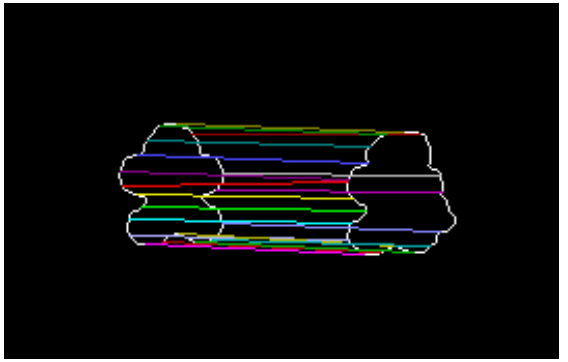












# Shape Finding in over-segmented images

Exploring using the partial shape matcher and a local aggregated search pattern within a global search pattern to find an object by shape alone.

The over segmented labeled regions tend to be a large enough number that thorough combinations of labeled regions within a total object size range is often infeasible.

Filtering the labeled regions by information that is thought to be conserved between images can be done to filter the regions.
A couple of the tests use HSV and CIE color filters.

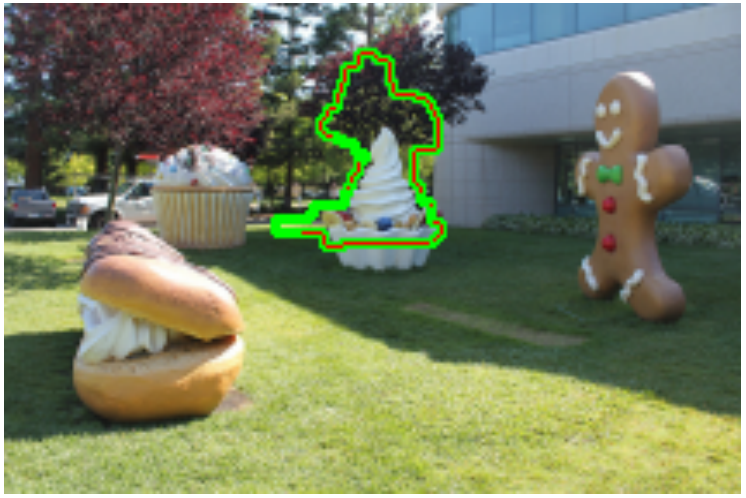
A texture filter can be built from the template object image (for each pyramid image), but this has a slightly large runtime (add complexity here).

The search using just shape on the filtered labeled regions has a long runtime complexity (add details here), and the true match is within the top results, but is not always the top result.

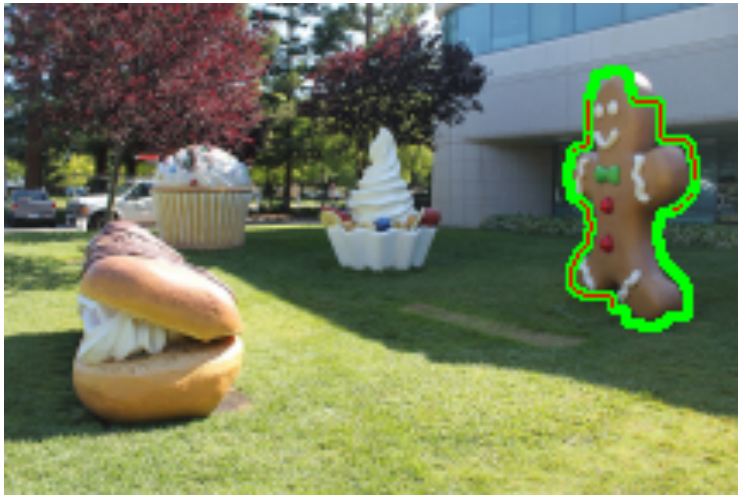
(a shape descriptor applied after top objects is a faster approach…see subsequent results)

# top matches to template shape within a test image

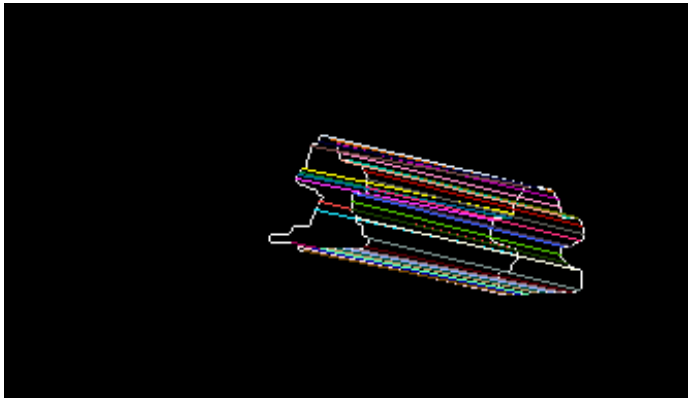
**best match** **and** **true match**

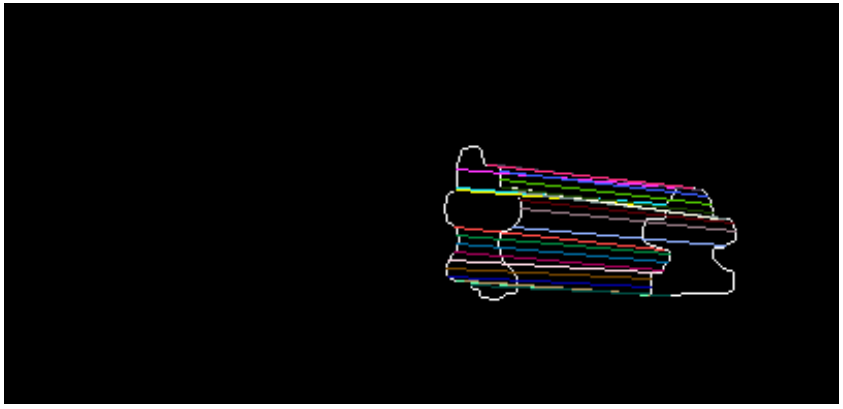


the min cost match to the template shape



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





# ORB key points and descriptors

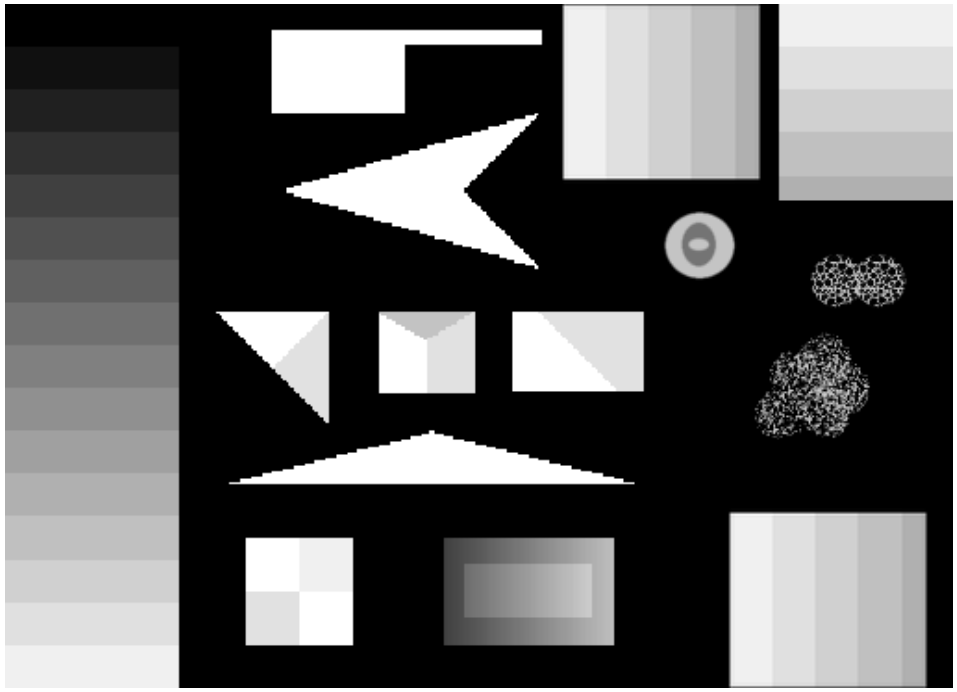
The key points for the template are at the left and on the right are key points filtered to those present in the segmented cells of the image to search, those cells were filtered by color histogram intersections with the template gingerbread man.

# texture

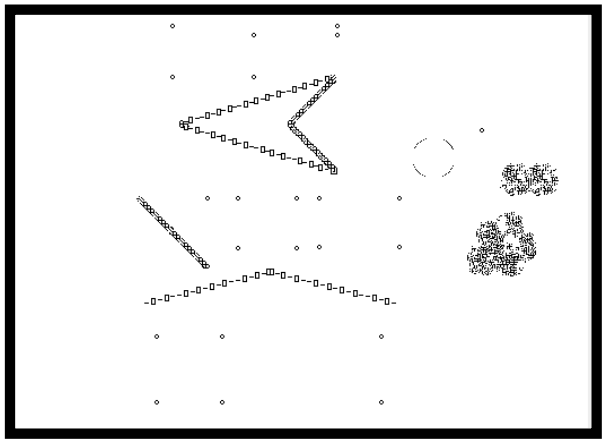
The highly textured regions such as vegetation produce many key points and those need to be filtered out for this case.
The Law's 1980 texture transforms (combinations of gaussian derivative convolutions) show that R5R5 may be a good way to find regions of high spatial and intensity variability, that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

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

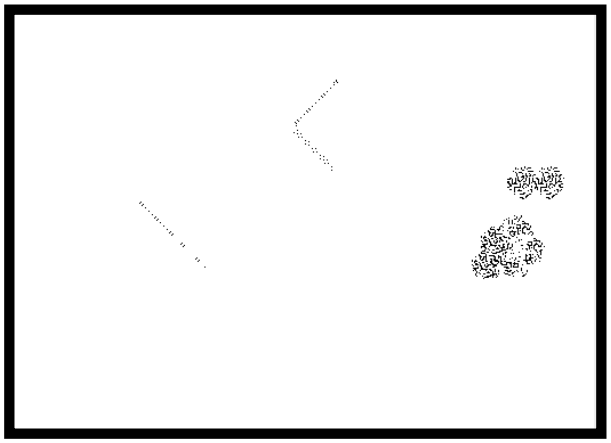
original image



E5E5 then adap med



R5R5 then adap med



# Shape Finding in over-segmented images (cont)

Finding the object (gingerbread man) in another image where it has changed location, lighting and pose requires very local and grouped information because the number of possible true key point matches compared to the total number of key points in the image is small.

**one approach in progress:**

(1) filter key points and segmentation in the image to be searched by filters of difference from the template object's colors (HSV and CIE LAB usually)

(2) filter the segmentation in the image to be searched to remove labeled regions with a higher surface density of key points after the region statistics have been scaled using size arguments. (substitution for a texture filter)

(3) for each labeled region, calculate euclidean transformation using point pair combinations and evaluate the pairs using all key points within a radius of template object size.

(4) using the octave combination and the labeled region's best transformation, include all the bounds of the matched key points' regions in a transformed nearest neighbor match within tolerance

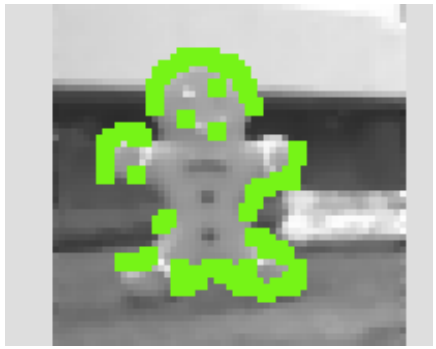
# Shape Finding in over-segmented images (cont)

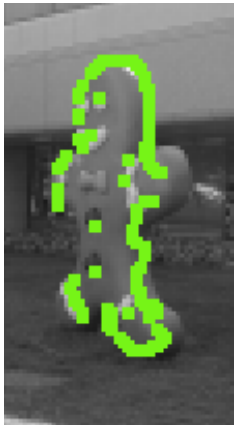
(5) for the octave and labeled region best result, used the matched boundary correspondence to calc chord difference costs within tolerance
(6) determined the cost of those “descriptors”, that is key point hsv costs, euclidean transformation distance costs, and chord difference costs to find the best results, then those within a percentage of best if any.

The best result at this point for one test is not yet the true match, but the 2nd best result is the true match.

*Further information is needed to determine that the true match is better.*

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

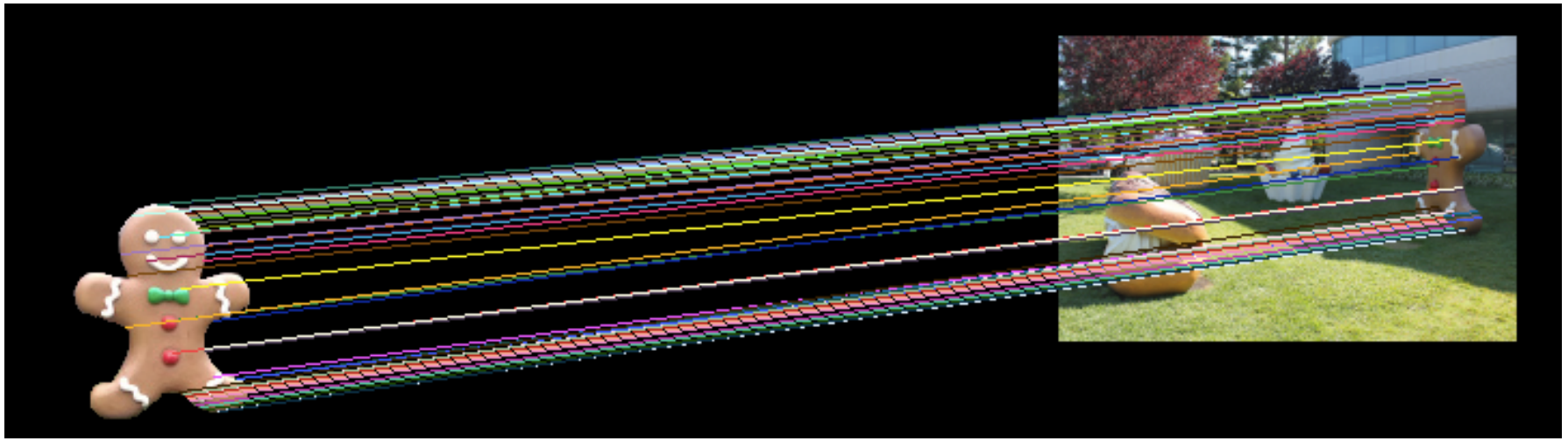




# Shape Finding in over-segmented images (cont)

(7) using the combined labeled region as contiguous small hsv descriptors to calculate the pixel by pixel similarity between transformed frame 2 and frame`1 pixels.

**true match is found**, at least for the test w/ largest projection effects



but alot of time is spent in pairwise euclidean transformations and evaluations

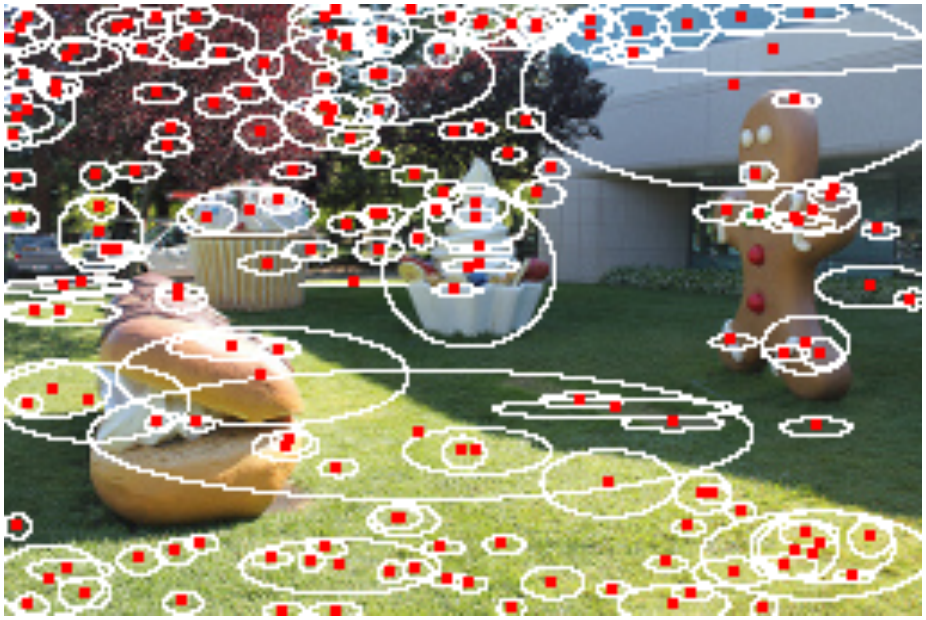
# Maximally Stable Extremal Regions (MSER)

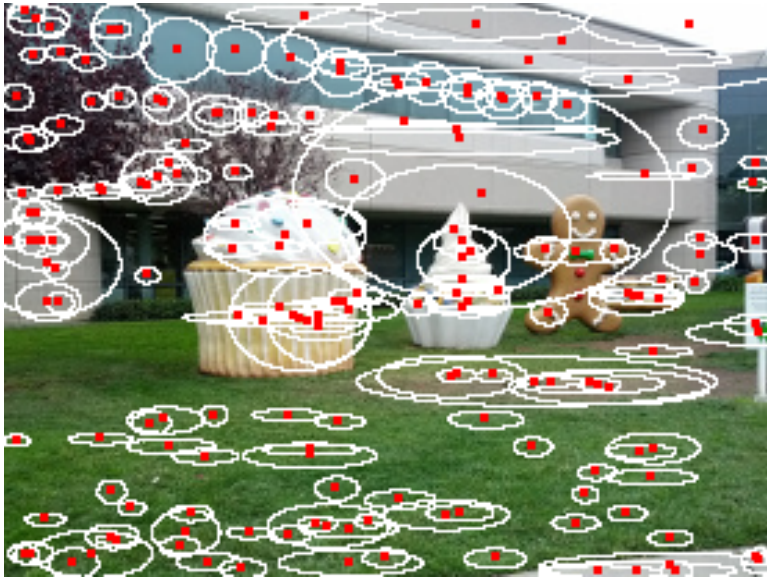
blob detection with MSER followed by canonicalization and descriptor matching is in progress.

*plotted below and on the next page are the MSER regions found on the inverted images.  The other sets of regions derived from the images are not plotted here.*

*especially useful for corner-less objects*

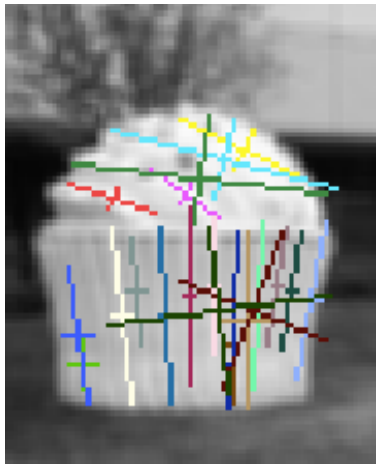
# MSER (cont.)

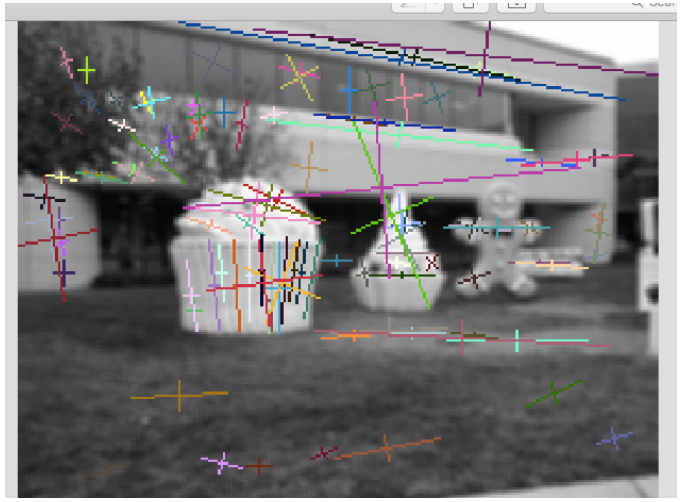


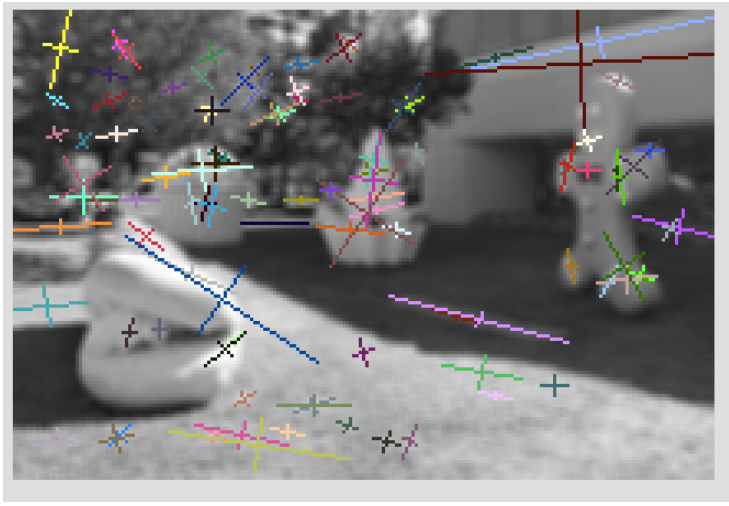


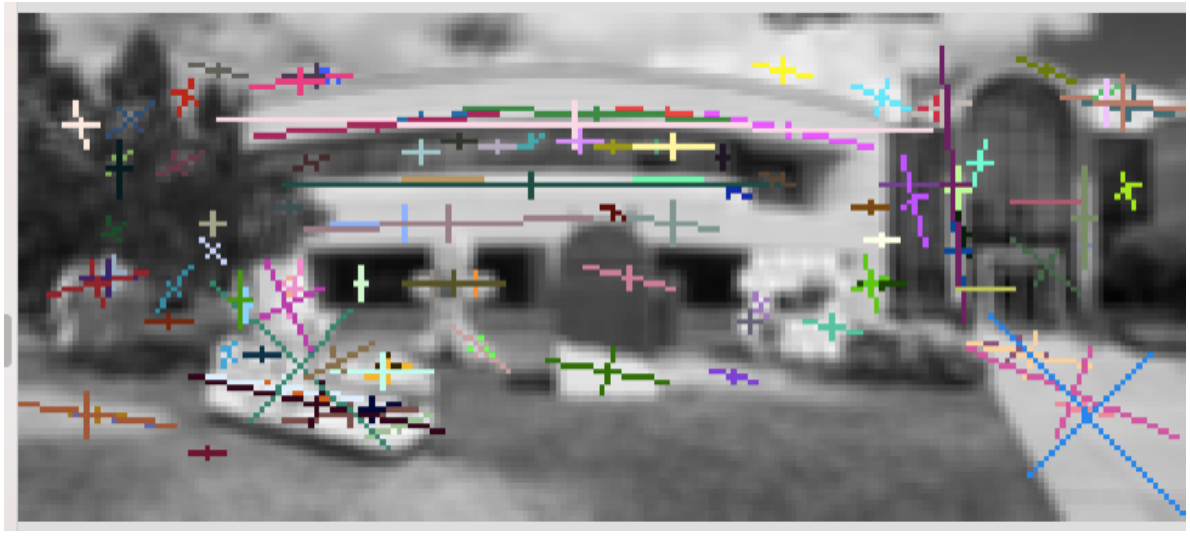
# MSER (cont.)

for the cupcake, there is extreme repetition of a pattern that if present in the searchable image, would help to further filter matchable mser regions.

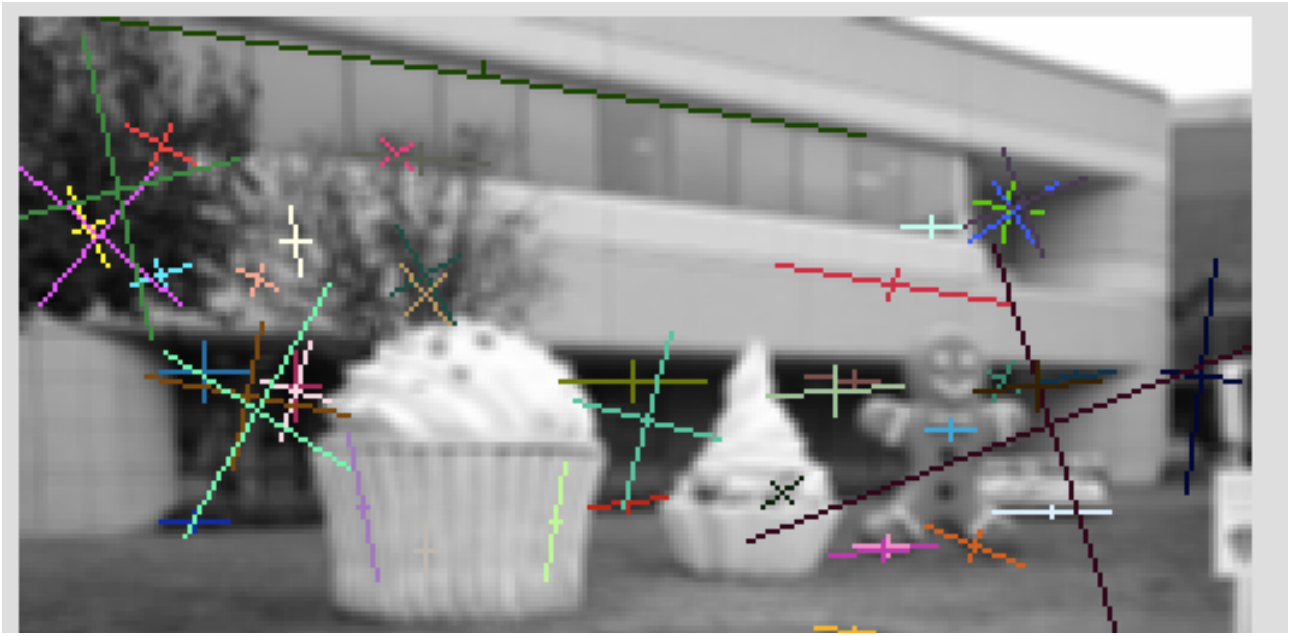


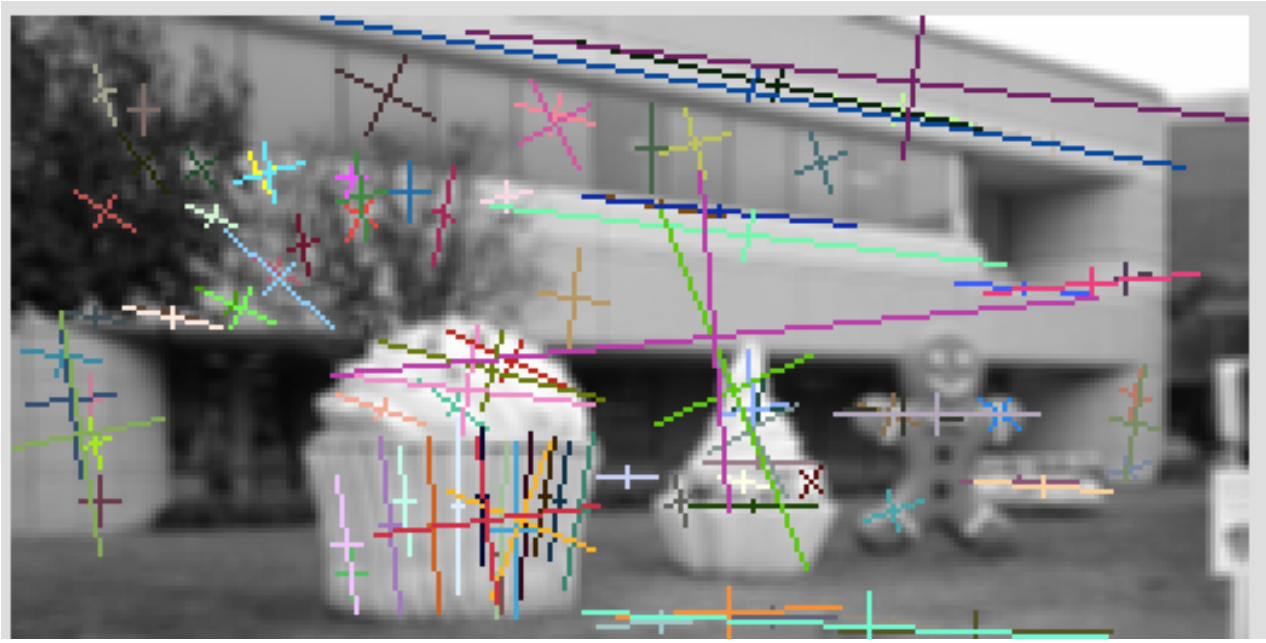






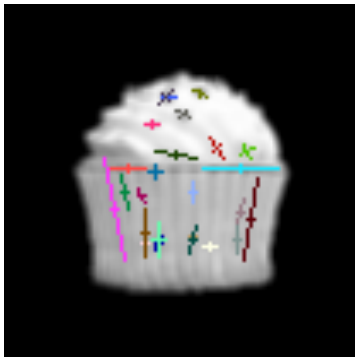
# MSER (cont.)





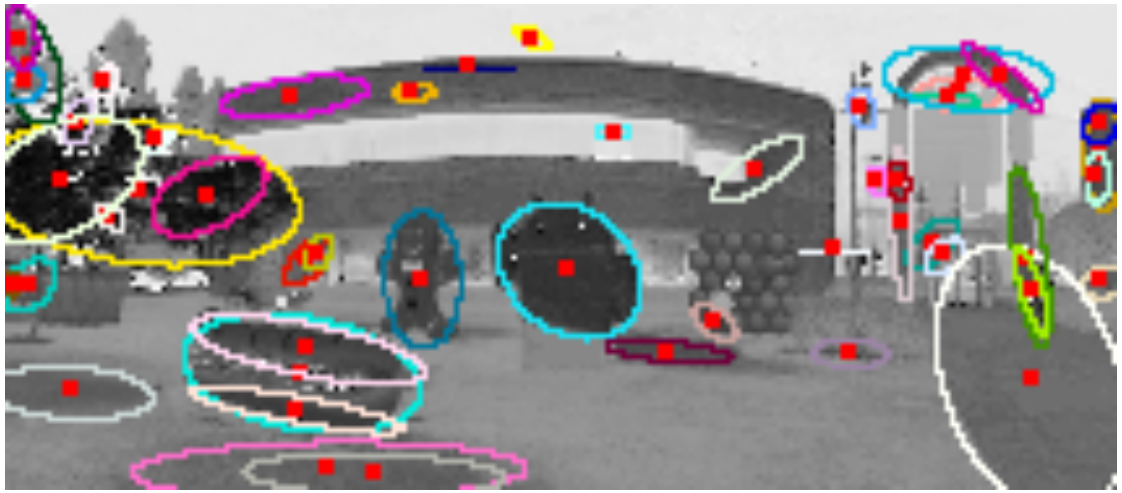
**By the way, the MSER semi-major axes possibly are very fast ways to find lines in an image.**

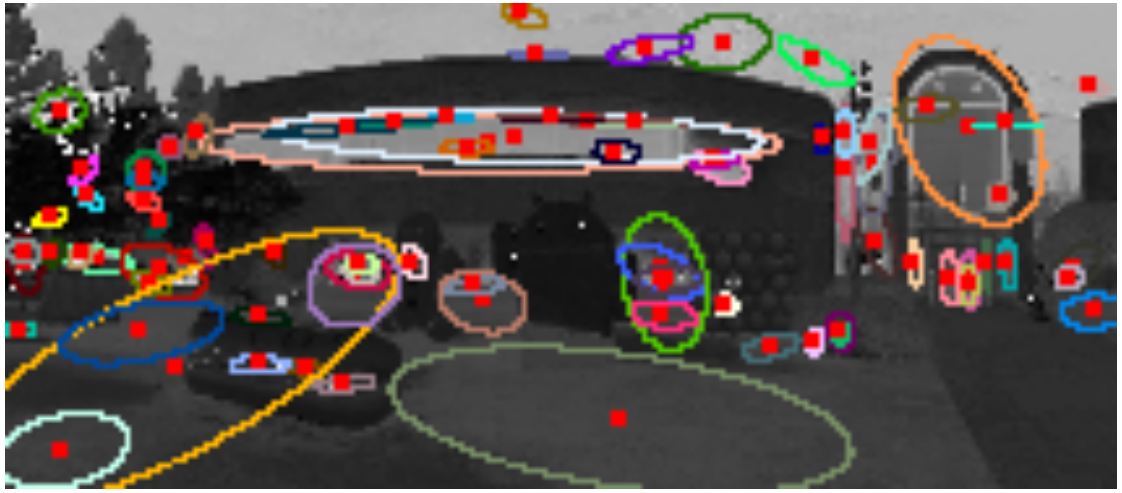
(imgs were filtered)



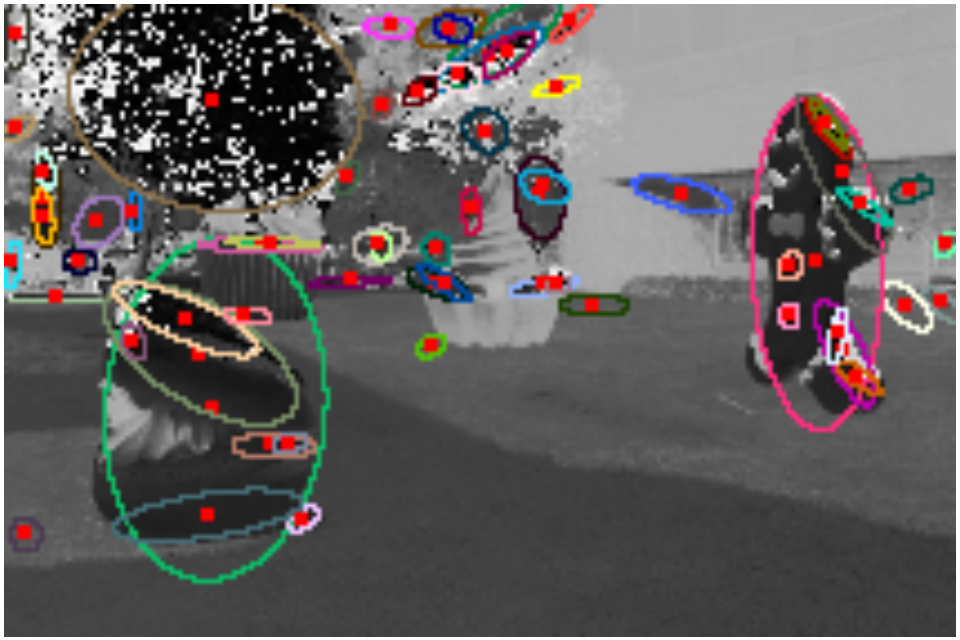
# MSER (cont.)

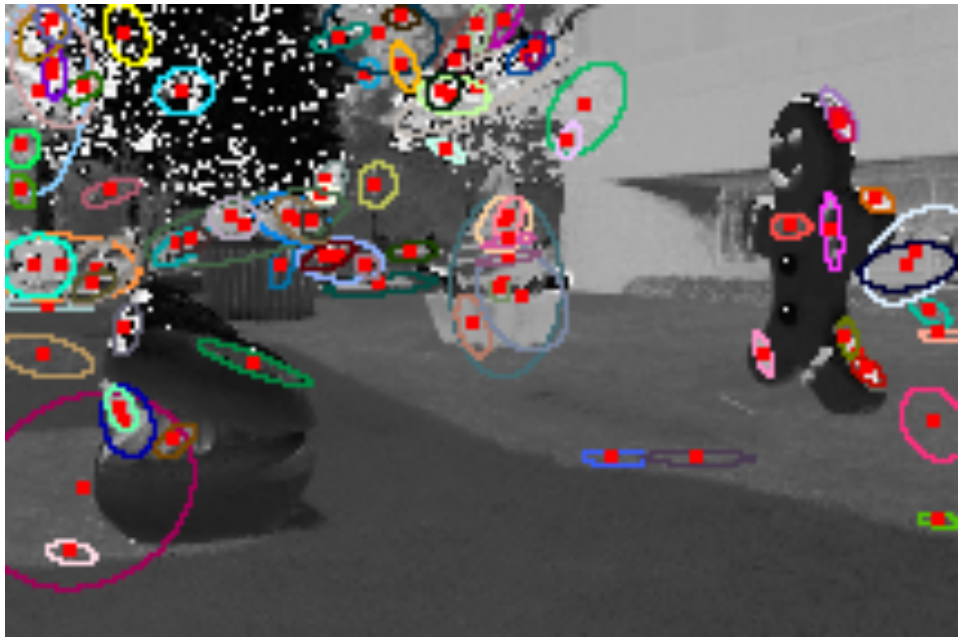
further looking at MSER applied to CIELUV polar theta images.
The MSER blob detector is finding objects and detailed regions.



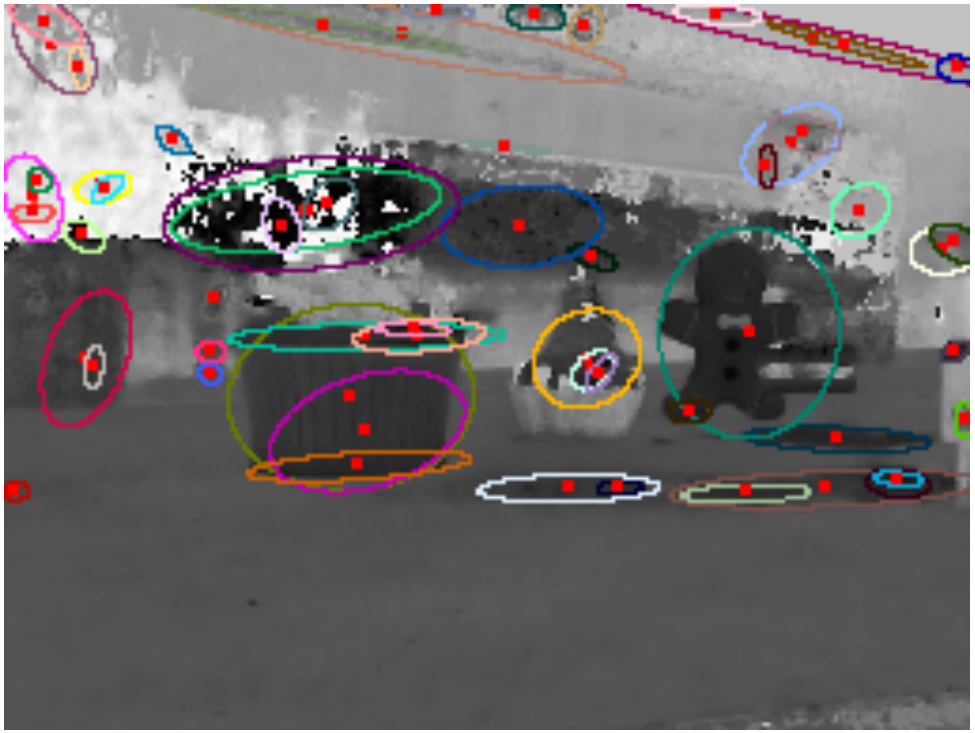


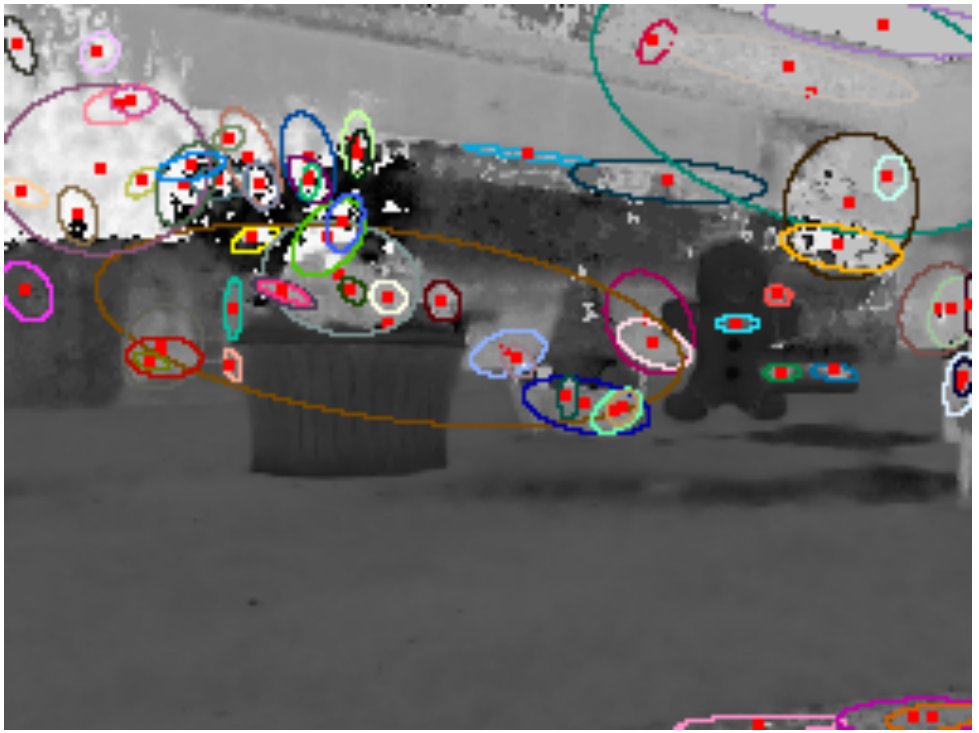
# MSER (cont.)



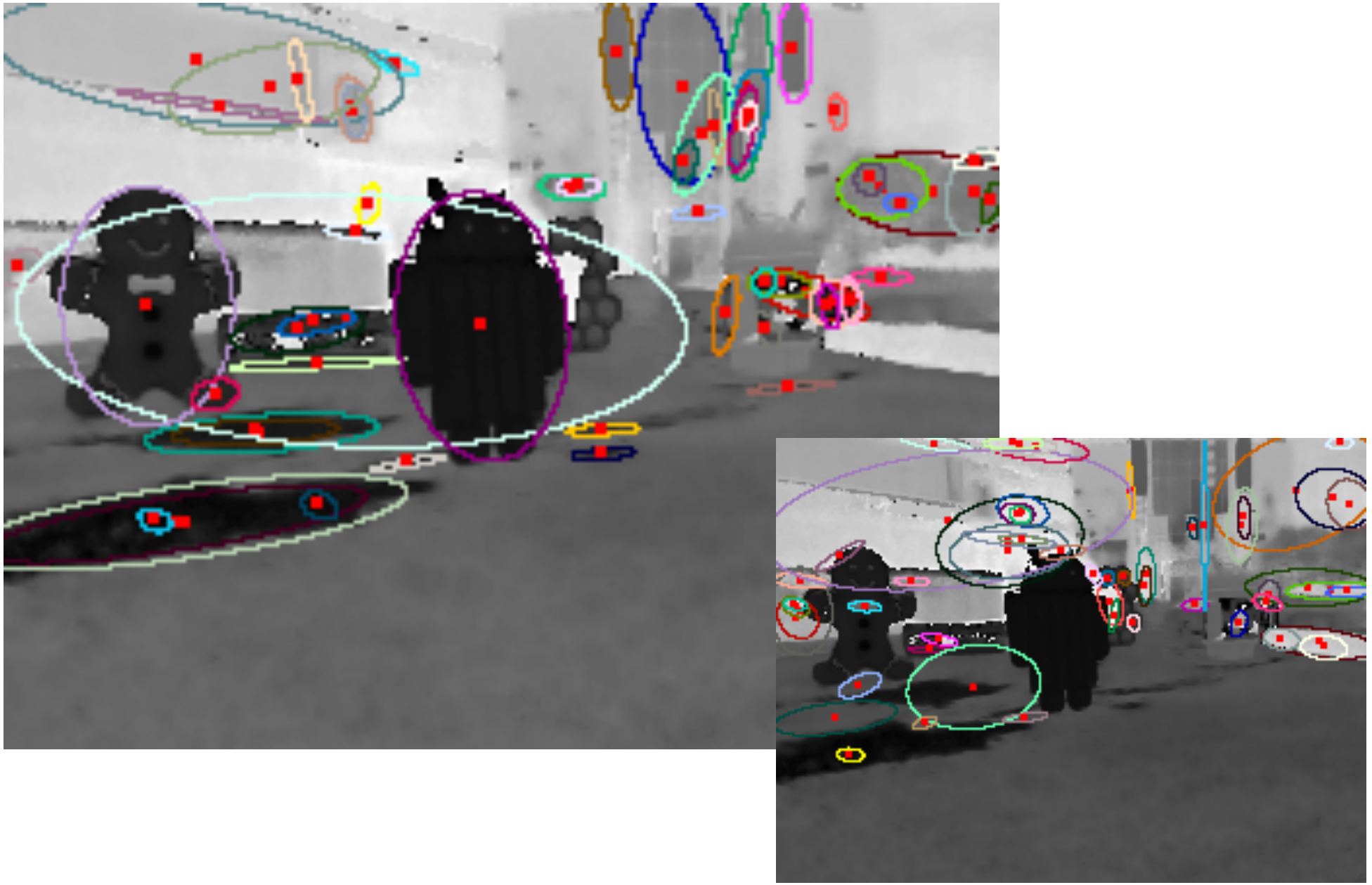


# MSER (cont.)





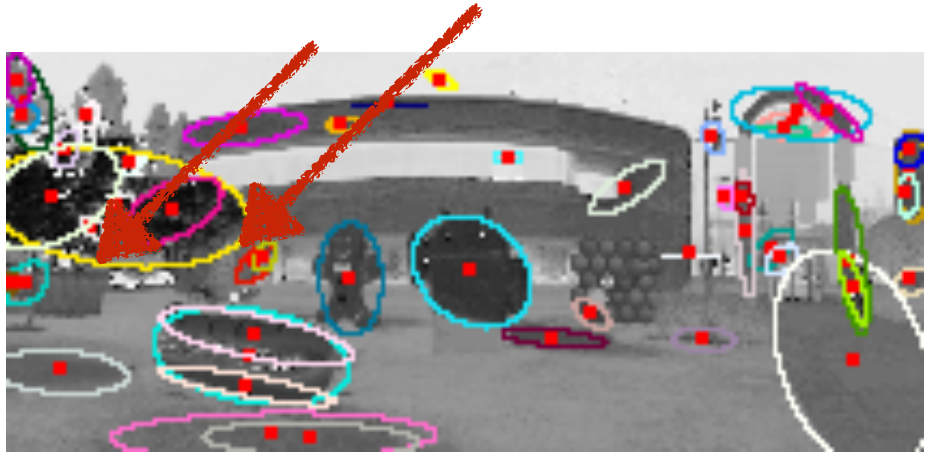
# MSER (cont.)

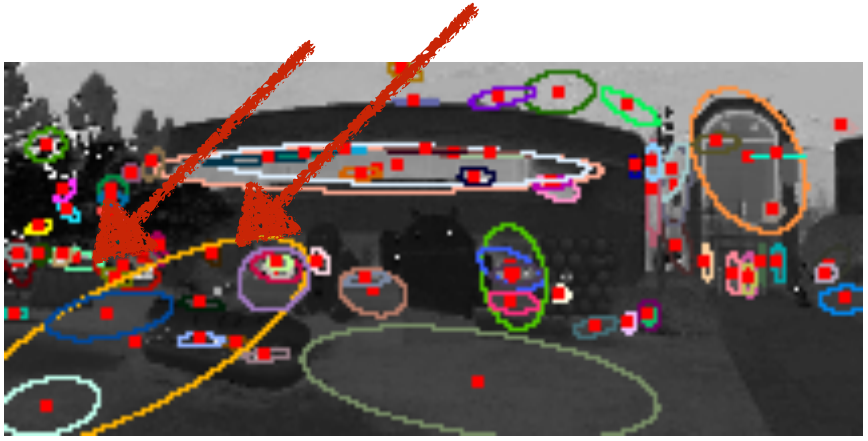


# MSER (cont.)

Using MSER on CIELUV polar theta images succeeds for most of the objects, but *fails for the lower resolution* android_statues_01.jpg in finding the *ice-cream and the cupcake*.

For those 2 objects, the intensity contrast is needed (see the previous MSER results).





# MSER (cont.)

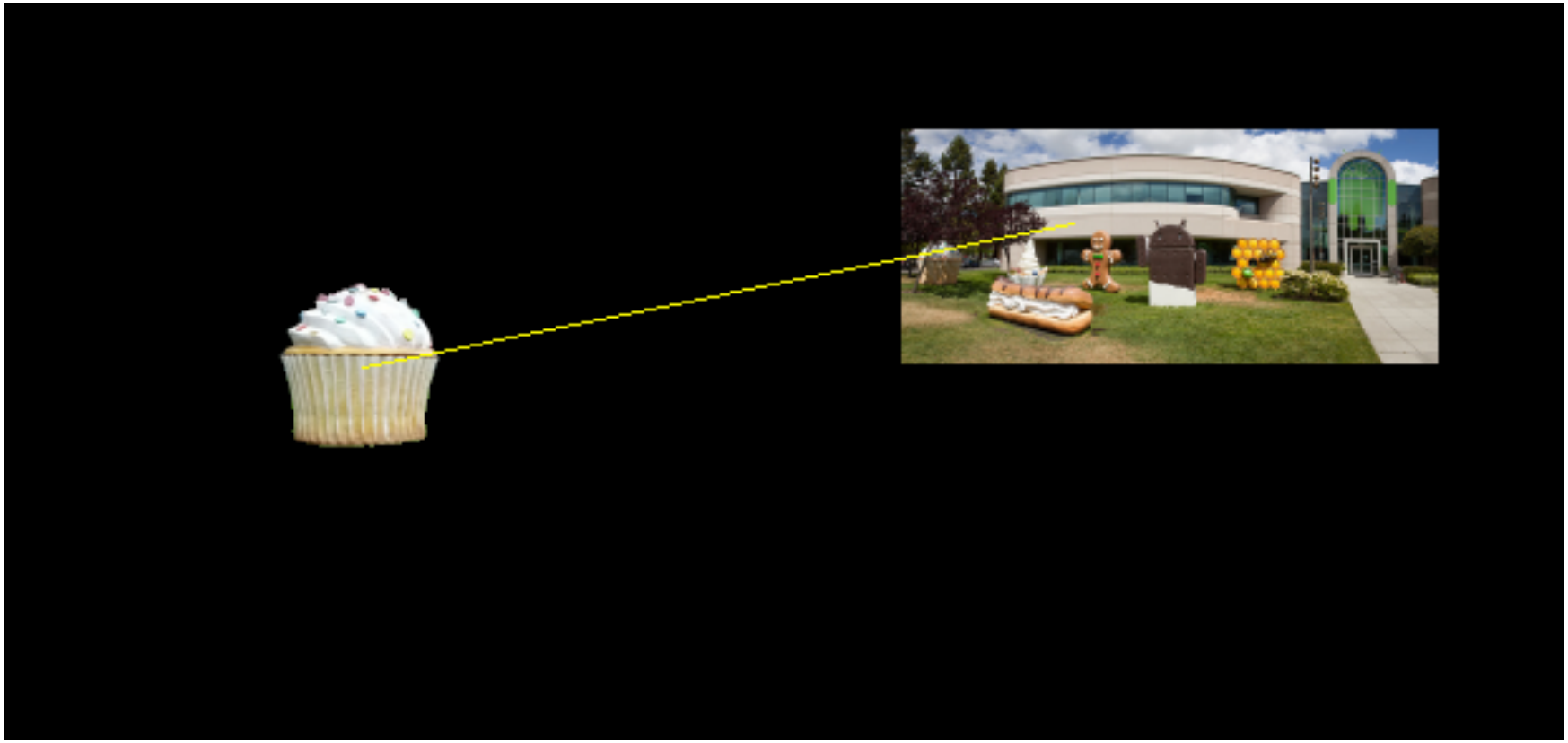
Finding the object (gingerbread man) in another image where it has changed location, lighting and pose, requires very local and grouped information because the number of possible true key point matches compared to the total number of key points in the image is small.

**the general approach:**

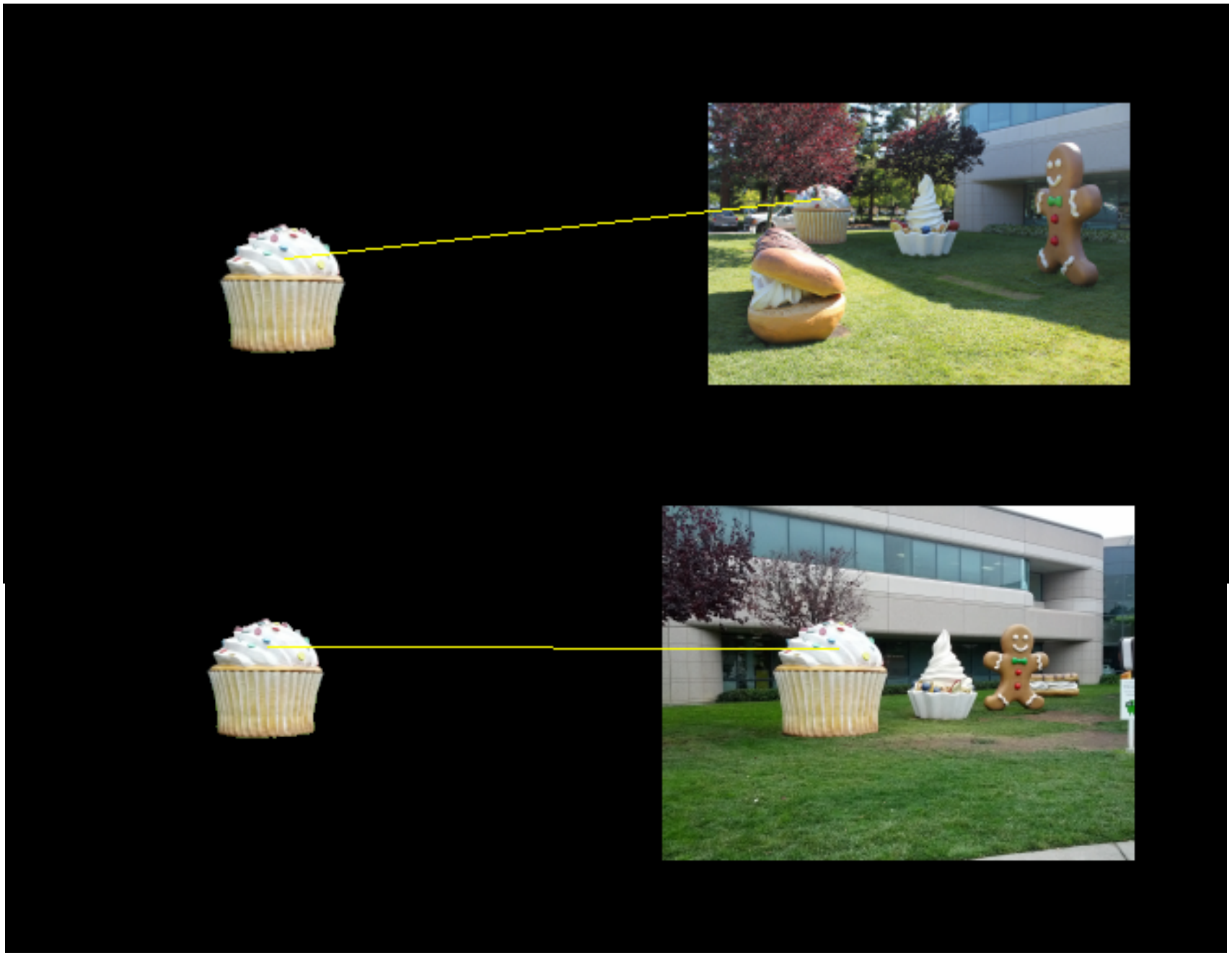
(1) Build an RGB pyramid for both images and calculate MSER regions for the largest image in both pyramids.
(2) Build a Polar theta CIRLUV pyramid for both images and calculate MSER regions for the largest image in both pyramids.
(3) filter the regions by color
(4) for each octave, find the best match of dataset0 objects (== template) to dataset1 objects and collect the best for each octave.
(5) Sort the best per octave by cost and then walk down the list if the HOG cost component is smaller for 2nd best to replace the top of best overall.

# MSER (cont.)

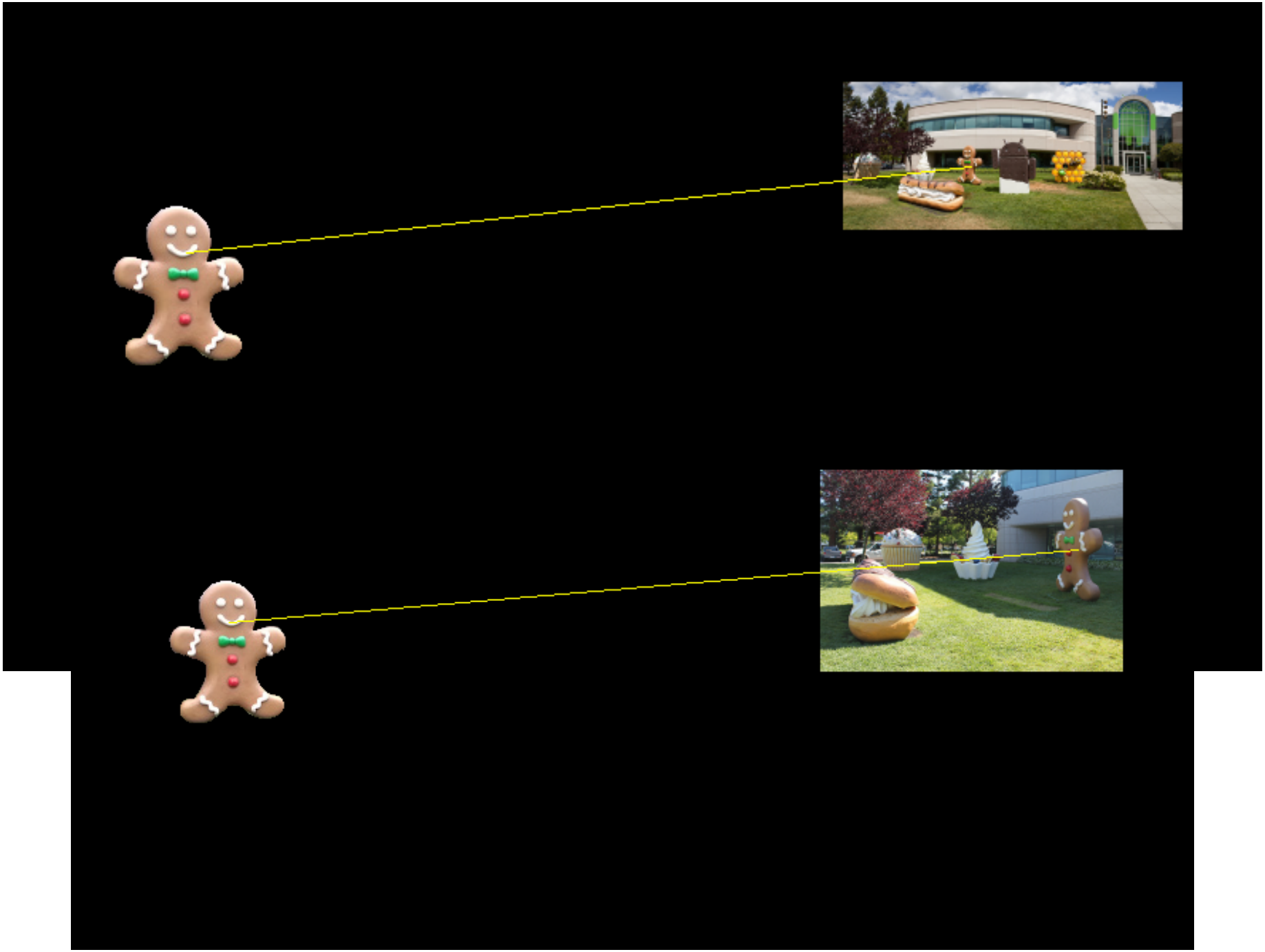
the android statue was found in 12 out of 13 tests.  The last one needs shape or other information to further distinguish it.



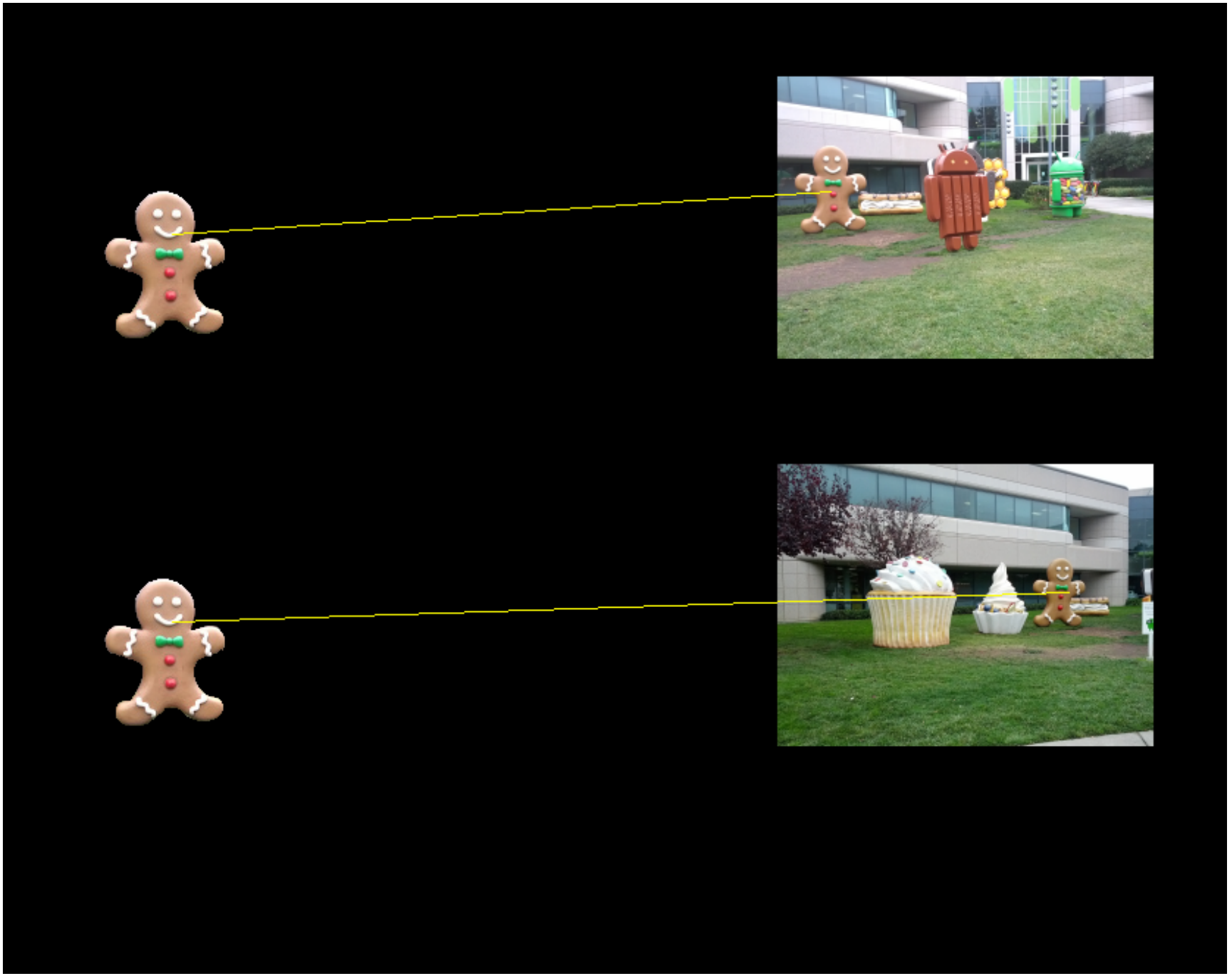
# MSER (cont.)



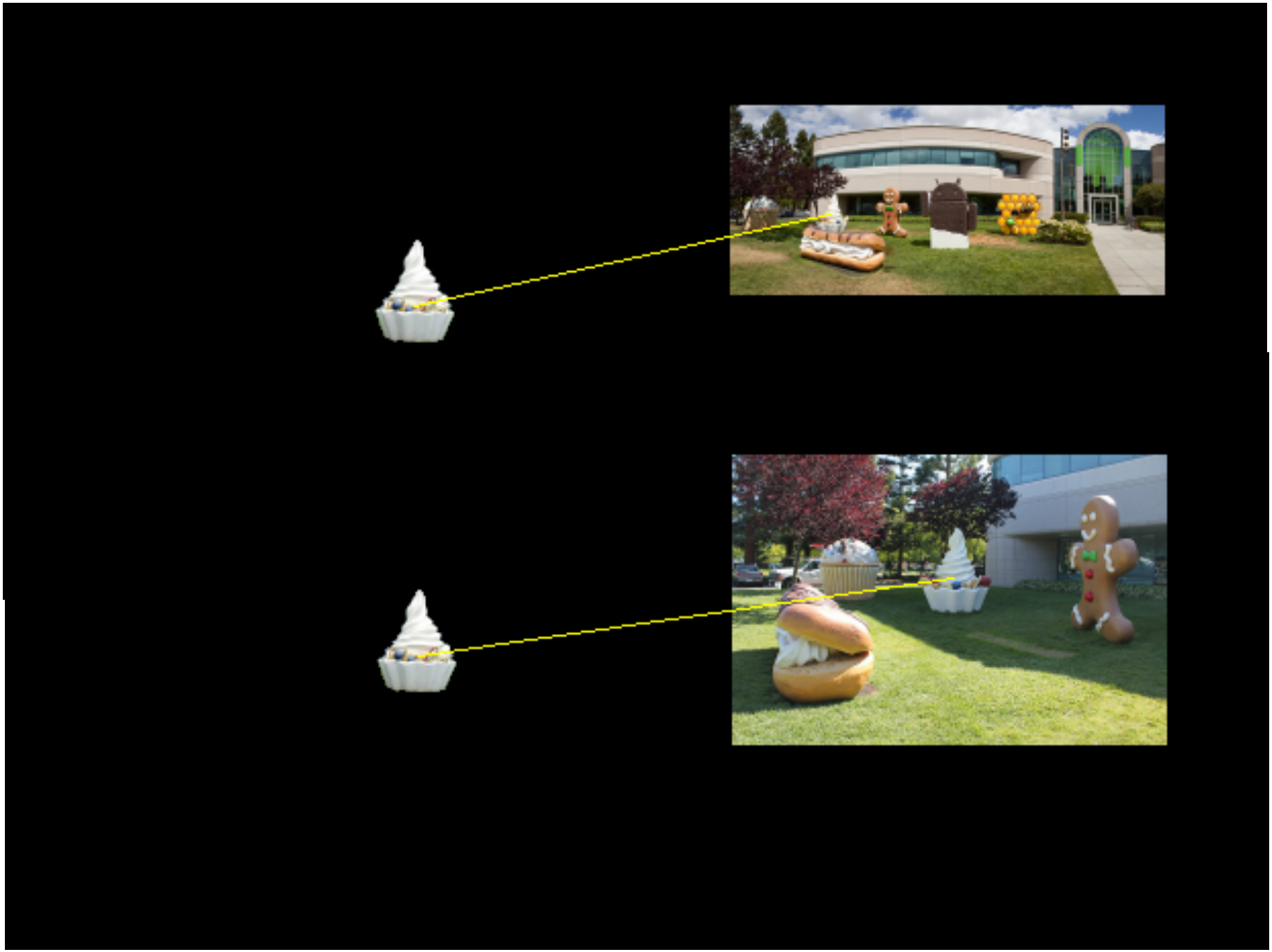
# MSER (cont.)



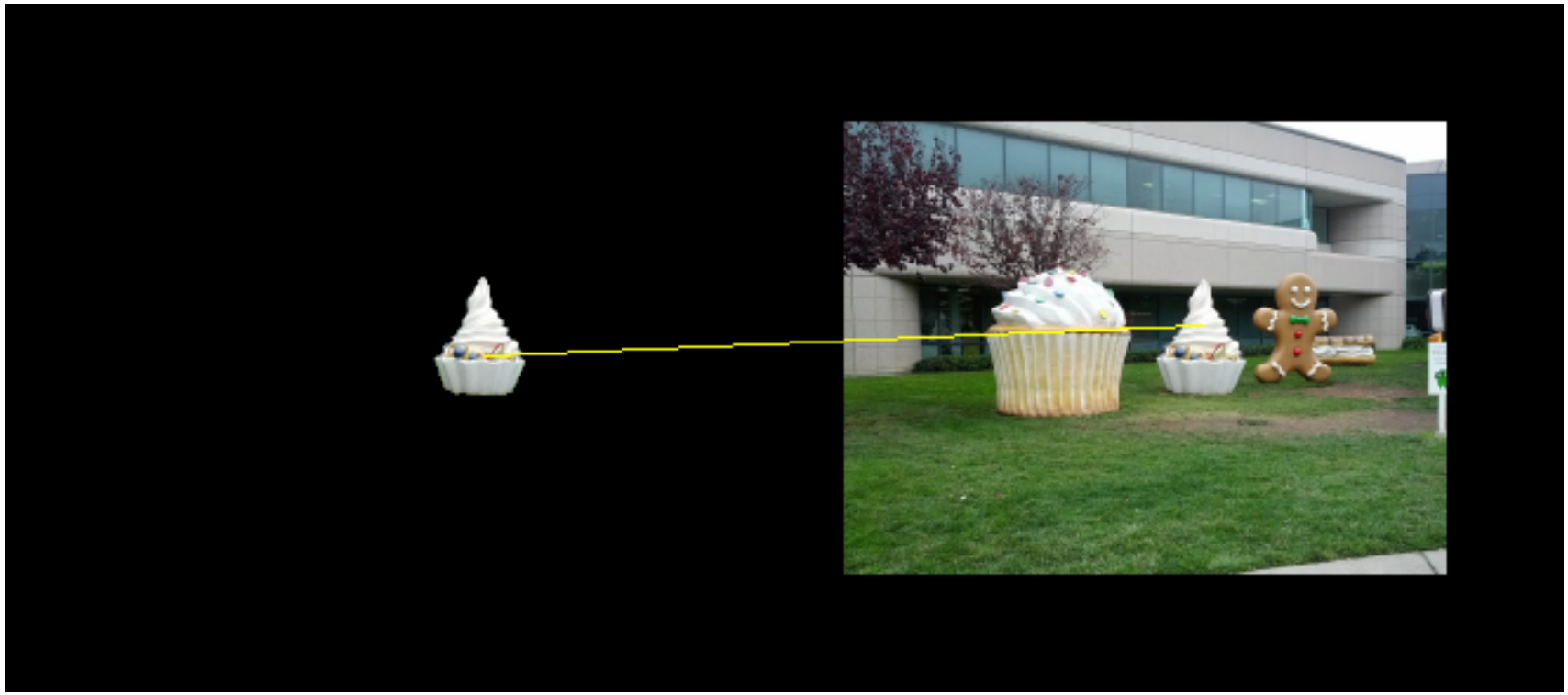
# MSER (cont.)



# MSER (cont.)

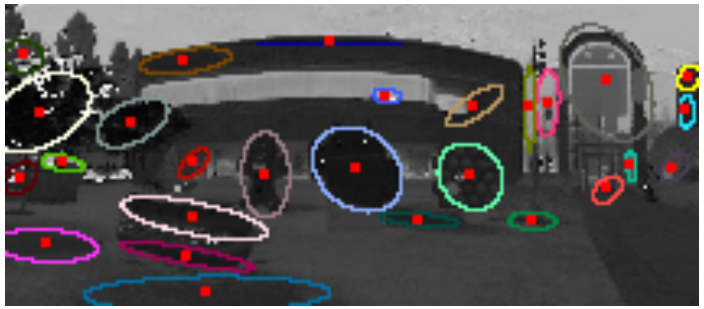


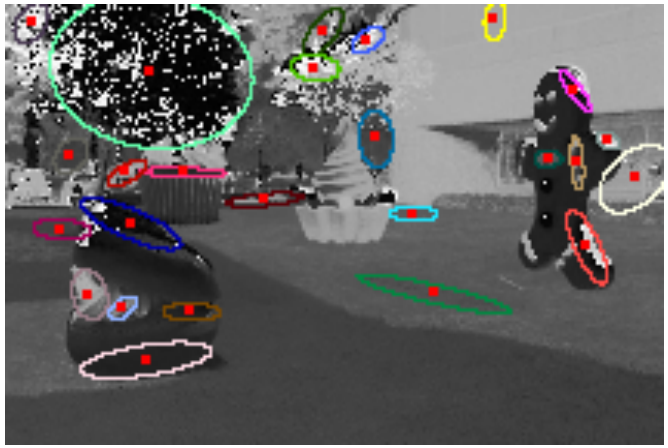
# MSER (cont.)



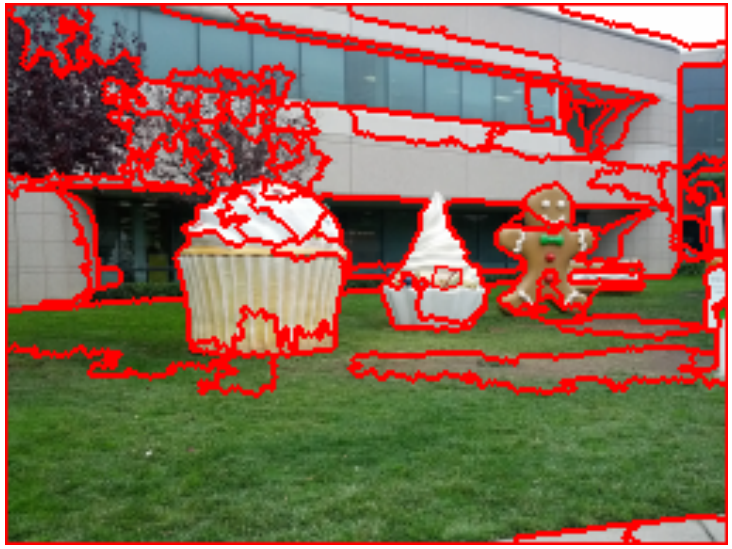
# MSEREdges (cont.)

created  a class called MSEREdges to make edges with the MSER regions created from greyscale and polar that CIE LUV images.
then a moderate amount of merging using color limits.

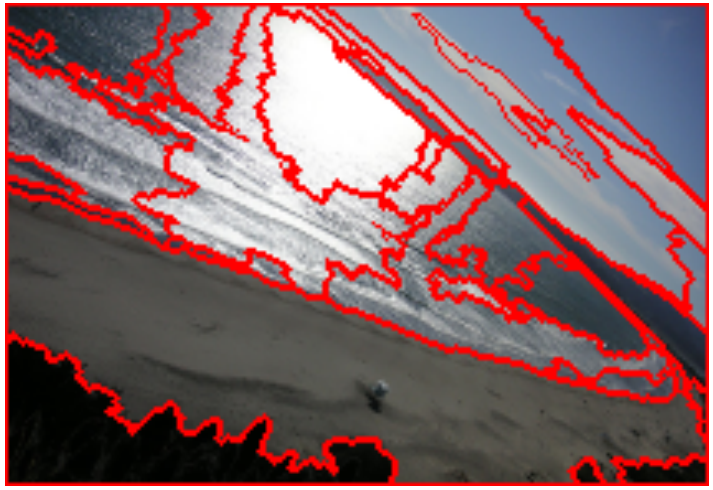




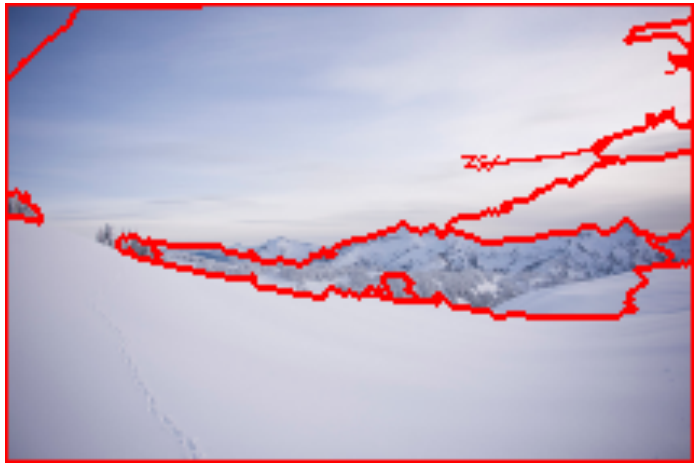
# MSEREdges (cont.)



# MSEREdges (cont.)

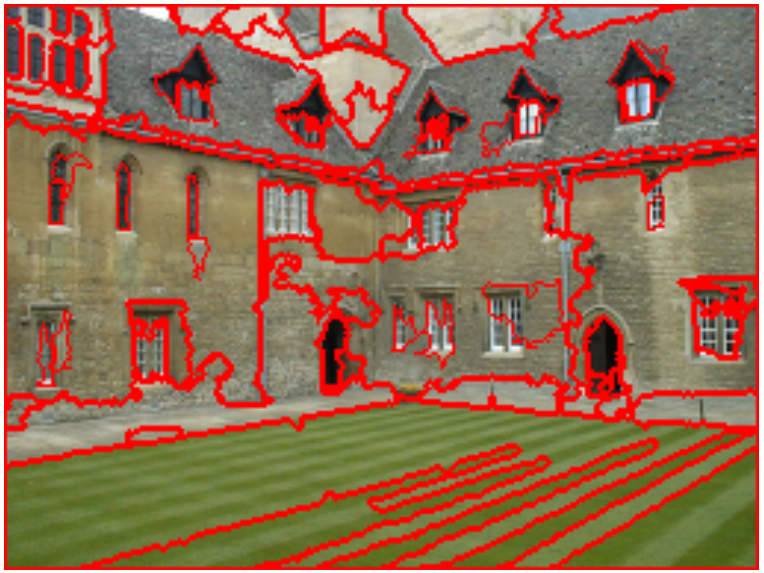


# MSEREdges (cont.)



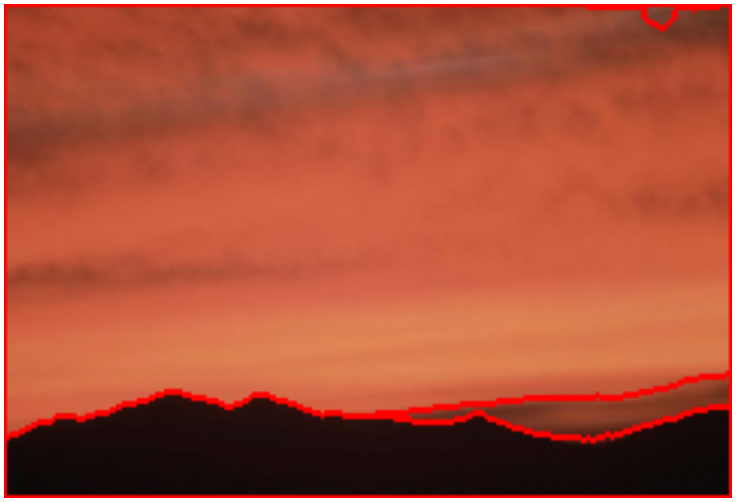
# MSEREdges (cont.)

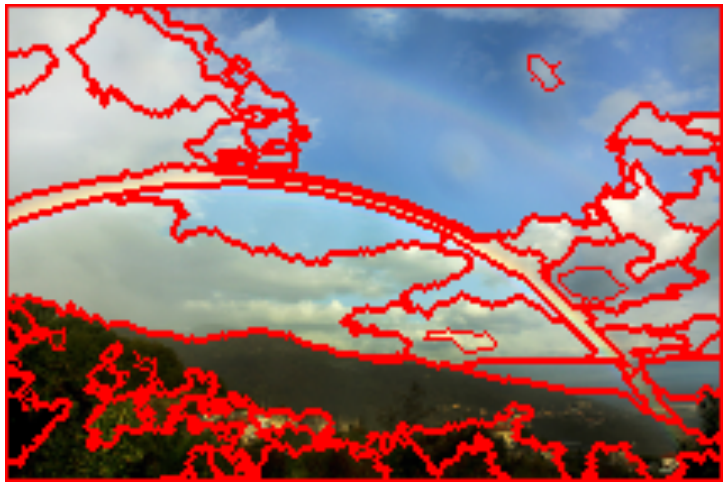
# MSEREdges (cont.)



# MSEREdges (cont.)

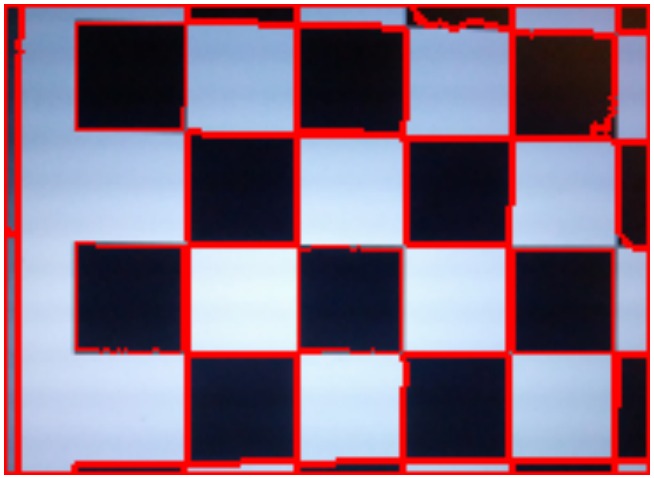
# MSEREdges (cont.)

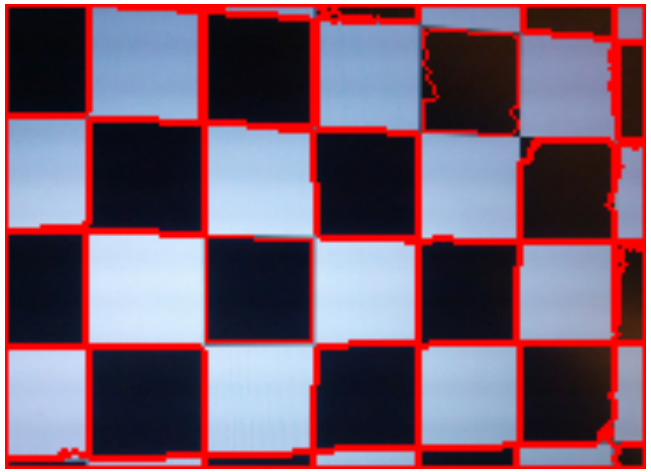




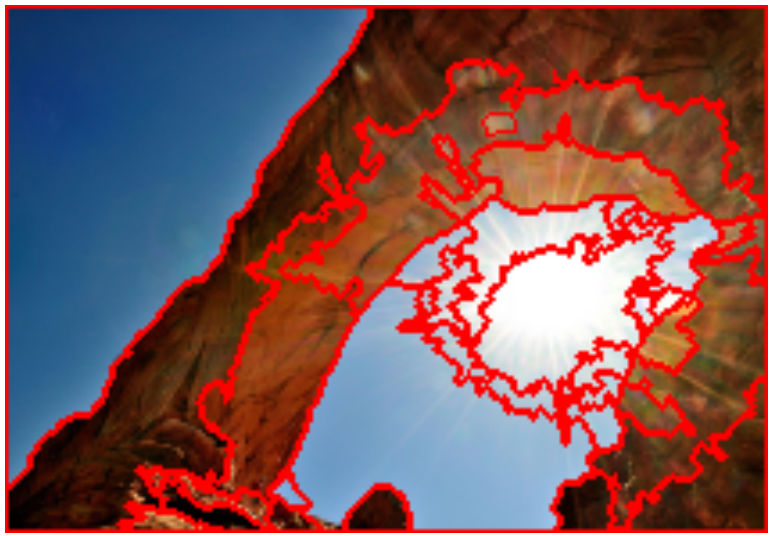
# MSEREdges (cont.)

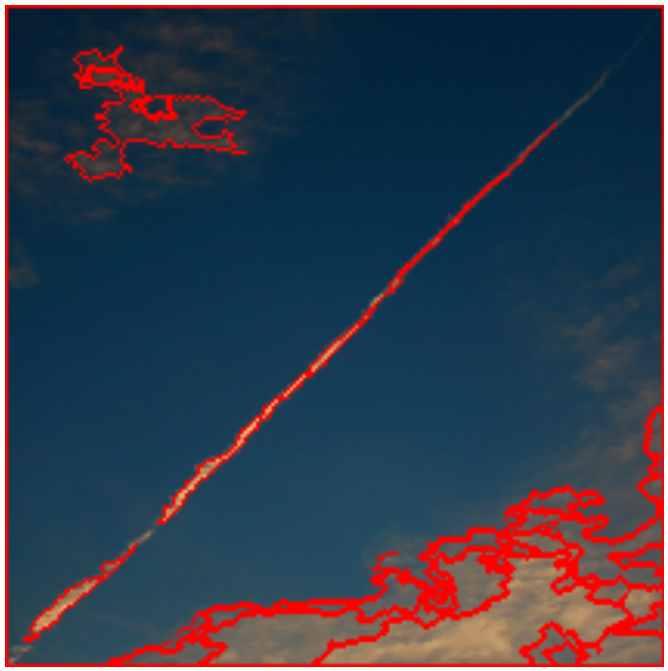






# MSEREdges (cont.)

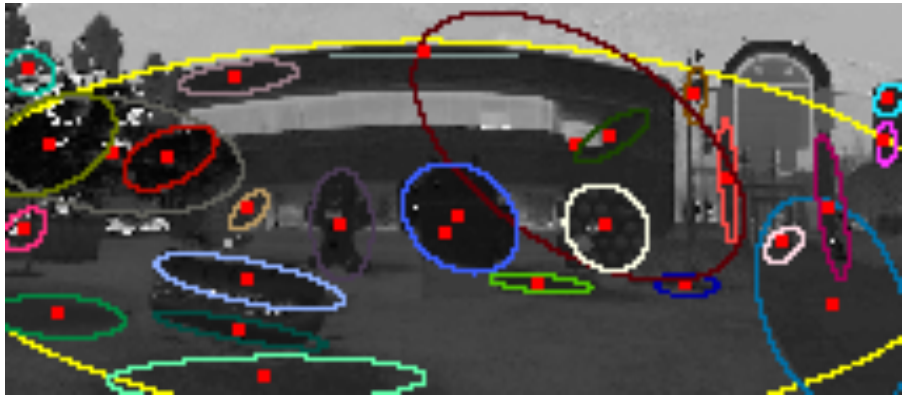




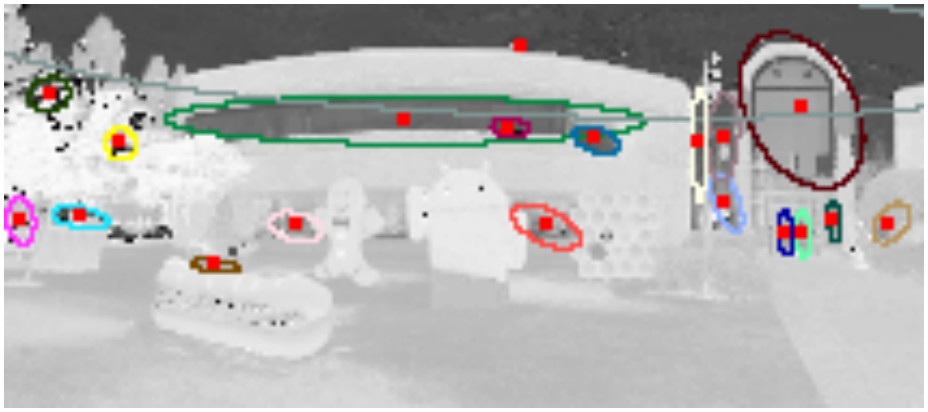
# MSEREdges (cont.)

correcting the polar theta image regions for wrap around effects

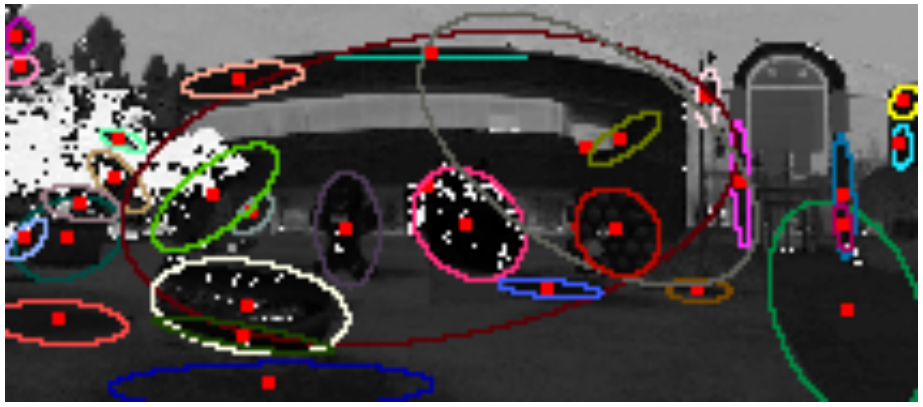
PT0



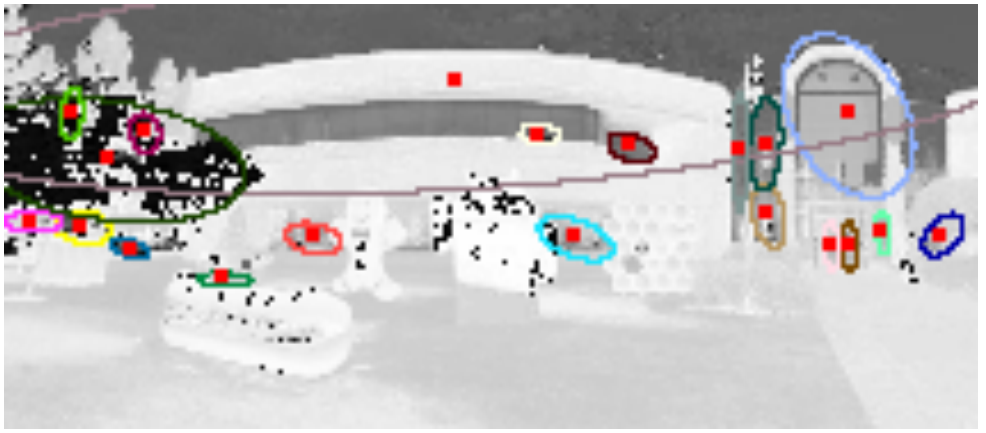
PT1



PT0 shifted by
20 degrees



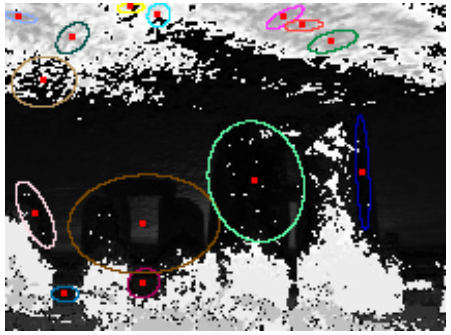
PT1 shifted by
20 degrees



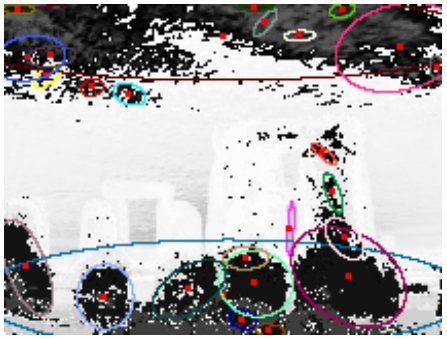
# MSEREdges (cont.)

correcting the polar theta image regions for wrap around effects

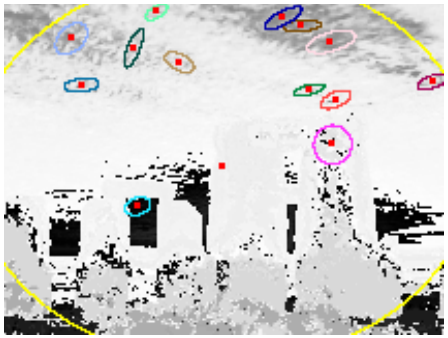
PT0



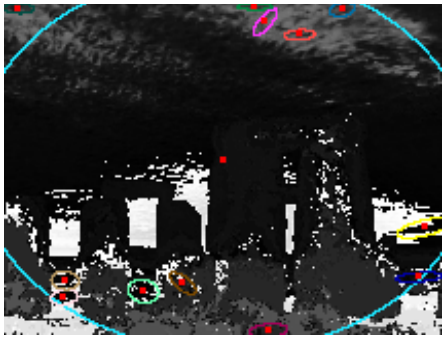
PT1



PT0 shifted by
20 degrees



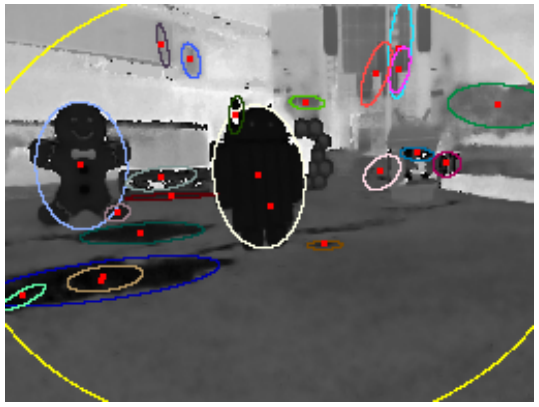
PT1 shifted by
20 degrees



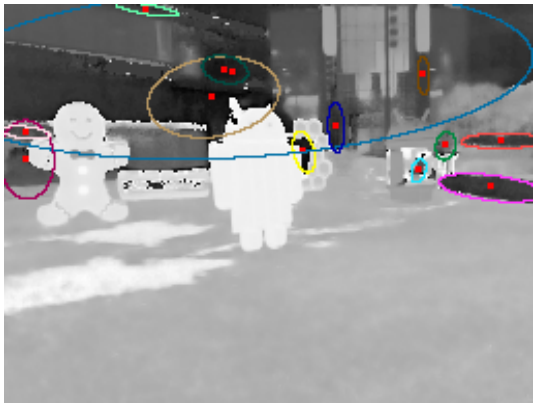
# MSEREdges (cont.)

correcting the polar theta image regions for wrap around effects

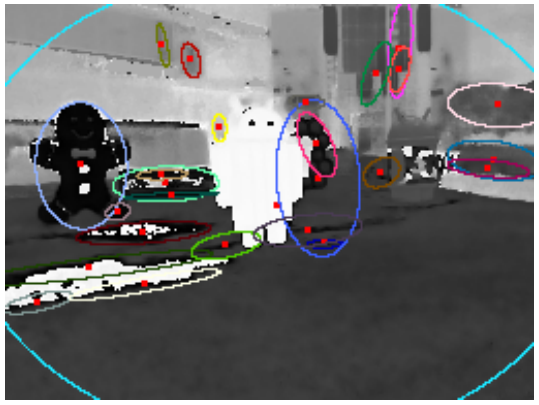
PT0



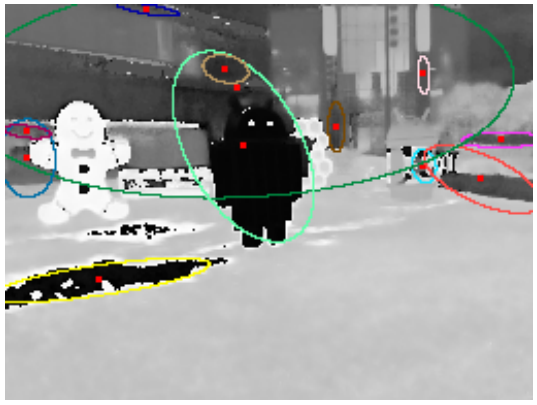
PT1



PT0 shifted by
20 degrees



PT1 shifted by
20 degrees



https://commons.wikimedia.org/wiki/File:HueScale.svg