

CS 5670: Computer Vision HW #1

Jeff Ponnor, Daniel Reidler

Task 1: Image Filtering and Enhancement

1. Output of Image Convolution:

	0	1	2	3	4	5	6
0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0
1	0	1.111111	2.555556	2.666667	2.888889	1.444444	0
2	0	1.888889	4.111111	4.333333	4.777778	2.555556	0
3	0	1.888889	4.222222	4.777778	4.888889	2.555556	0
4	0	1.444444	3.666667	4.222222	4.666667	2.444444	0
5	0	0.666667	2.111111	2.555556	2.777778	1.333333	0
6	0	0.000000	0.000000	0.000000	0.000000	0.000000	0

2. Output of Convolution With Median Filter:

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	2	1	2	0	0
2	0	2	4	5	6	2	0
3	0	2	4	5	7	2	0
4	0	1	4	4	4	3	0
5	0	0	1	1	3	0	0
6	0	0	0	0	0	0	0

The key difference from 1:

	0	1	2	3	4	5	6
0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0
1	0	1.111111	0.555556	1.666667	0.888889	1.444444	0
2	0	-0.111111	0.111111	-0.666667	-1.222222	0.555556	0
3	0	-0.111111	0.222222	-0.222222	-2.111111	0.555556	0
4	0	0.444444	-0.333333	0.222222	0.666667	-0.555556	0
5	0	0.666667	1.111111	1.555556	-0.222222	1.333333	0
6	0	0.000000	0.000000	0.000000	0.000000	0.000000	0

3. Gradient Magnitude and Direction using Sobel:

Gradient Magnitude: 4.472136

Gradient Direction: -1.107149

4. a) Filter using distance between pixels: We use a simple Gaussian 3x3 kernel.

[0.0625,0.125,0.0625],

[0.125,.25,0.125],

[0.0625,0.125,0.0625], (Please see comments in code for more details.)

b) Filter using distance between pixel values:

-- For each 3x3 neighborhood, we order pixels values least to greatest. If center pixel value is the least or greatest in the neighborhood, it will weigh 6x and we will equally weigh the other 8 pixels. Else, the center pixel value is weighted 4x, Weight nearest pixel value below center pixel- 2x, Weight nearest pixel value above center pixel- 2x. In both cases, we weight the remaining pixels 1x. We set the center pixel value to weighted average of values.

c) Filter that takes into account both distances:

-- We take the average of the values computed in a and b.

Results:

Filter 1	Filter 2	Filter 3
		
		
		

5. Unsharp Masking:



The differences in the images

Cameraman -

1. The original grayscale image is the sharpest.
2. The image with sigma = 2.5 is "dull" in the sense that the image seems to be more gray and use a smaller range of values.
3. The image with sigma = 0.75 is not as sharp as the original but also still clearer than the image with sigma = 2.5

House

1. The original grayscale image is the least sharp of the images.

2. The image with $\sigma = 2.5$ is the sharpest. Specifically, the contrast between the bricks and the house gutter is more sharp.
3. image with $\sigma = 0.75$ is sharper than the original but not as sharp as $\sigma=2.5$

Lena

1. Like the cameraman, the original is the sharpest. Though, the image with $\sigma = 0.75$ is almost as sharp.
2. The image with $\sigma = 2.5$ is not as sharp.

Task 2: Color Quantization with K-Means

1. Original Image:

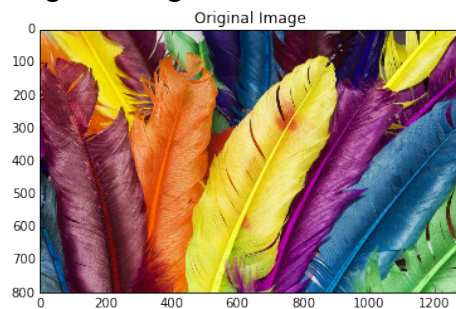
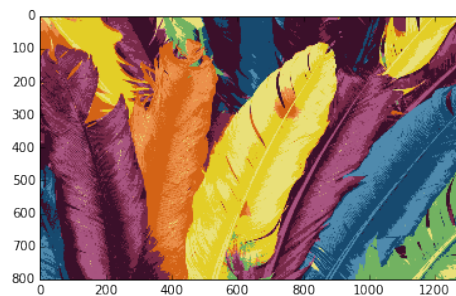
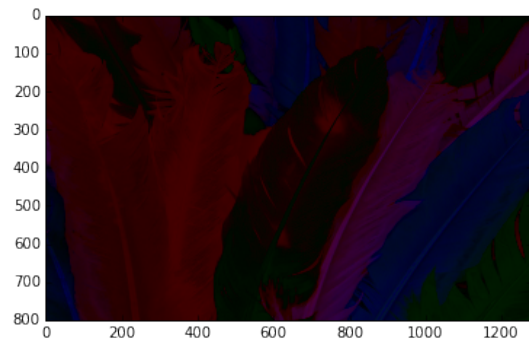


Image after Quantization:



2. Image after Quantization of L-channel:



3. Sum of Squared Distances (Tested on the same image, should equal zero)

3)SSD between two RGB images

```

In [82]: def ssd(img1,img2):
          sum = 0
          for i in range(len(img1)):
              for j in range(len(img1[i])):
                  for k in range(len(img1[i][j])):
                      sum = sum + ((img1[i][j][k]-img2[i][j][k])*(img1[i][j][k]-img2[i][j][k]))

          print "The sum of squared distances between the two images is ", sum, "."

          pass

ssd(mpicg.imread('colorful2.jpg'),mpimg.imread('colorful2.jpg'))

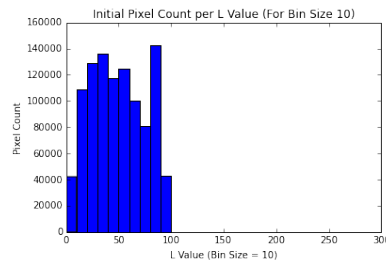
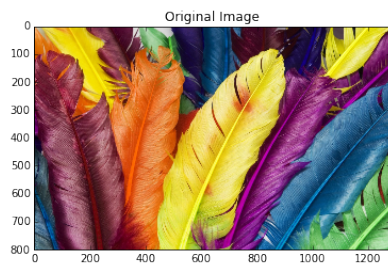
The sum of squared distances between the two images is 0 .

```

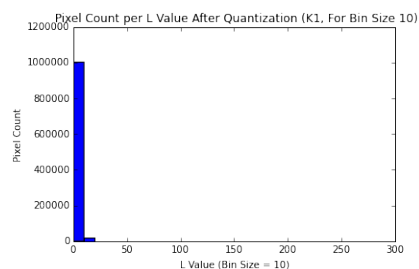
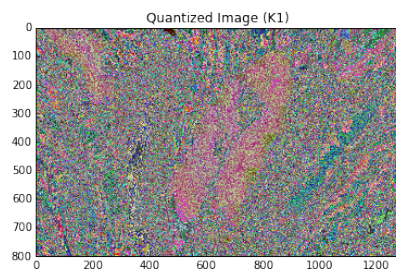
4. (Problem 5 contains the results of the Histogram Generation function since it is being called). The code takes the input image and the bin size as parameters. It then converts to LAB and goes through the L-channel counting the pixel values. Then matplotlib's histogram creation function is used to create the histogram. Then the quantization function written earlier is called and a new histogram is created from the L-channel values.

A) Results for Image 1:

5.

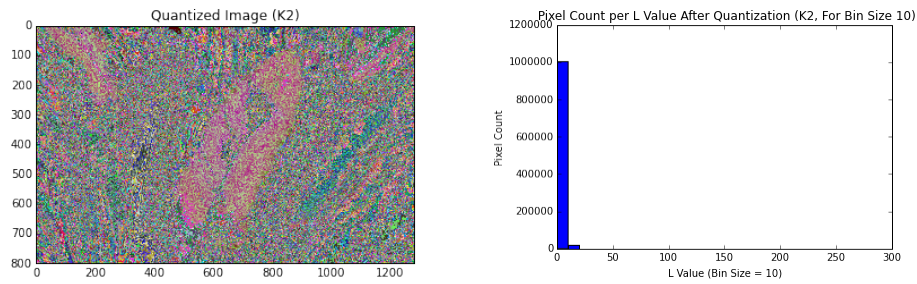


Quantization with K = 10



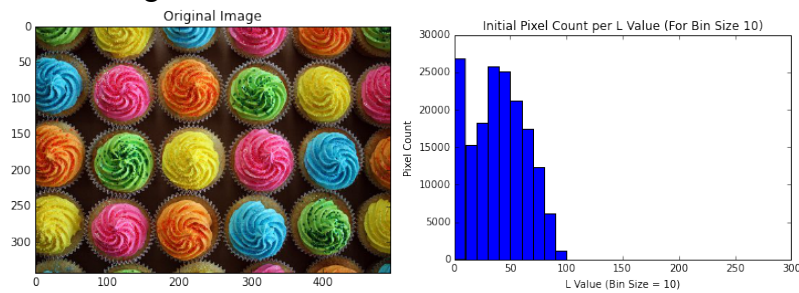
The sum of squared distances between the two images is 5589732187.08.

Quantization with $K = 20$

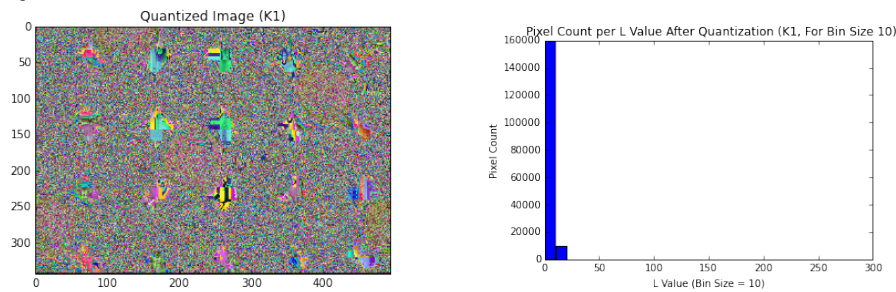


The sum of squared distances between the two images is 5589693057.68 .
The difference between the quantized image and original seems to be less than before when comparing the SSD.

B) Results for Image 2: The sum of squared distances between the two images is 5589732187.08 .

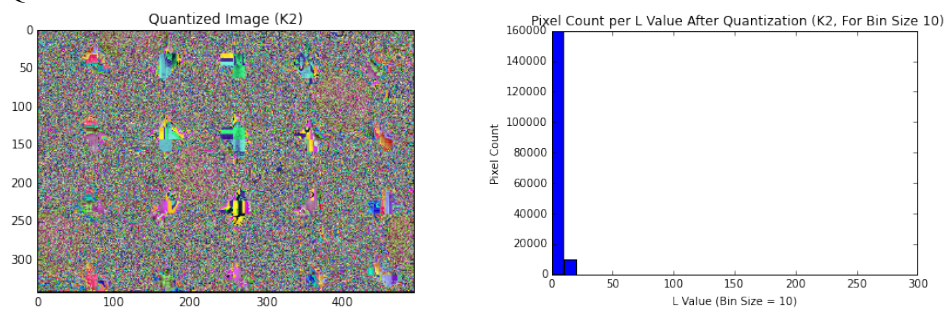


Quantization with $K = 10$



The sum of squared distances between the two images is 5589732187.08 .

Quantization with $K = 20$



The sum of squared distances between the two images is 5589693057.68 .

As with the previous image the difference between the quantized image and original seems to be less than before when comparing the SSD.

The results show how Quantization reduces the number of values (in case of LAB)/colors (in case of just RGB) in an image. The histograms show how the pixel count within a bin increases while the number bins decreases. This is expected as Quantization with K-Means essentially builds a model of the image with a limited set of data. The fact that the model is built from a limited set of information is what resulted in the reduction of the variety of values (depending on which format LAB or RGB). Increasing the K values improves the quantized image quality (closer to the original) and results in a histogram that is also closer to the original one. This is because when the number of clusters, K, is increased, the model has more data points to estimate from. It can differentiate between a greater variety of values and provide more detail. The Sum of Squared Distances also makes this clear since the difference between the original image and quantized image with many clusters is less than the difference between the original image and a quantized image with a few clusters. Running the program does not result in the same results every time. This is because the K-Means algorithm chooses points randomly to generate a model so there will be some differences between each round.

Task 3: Edge Detection

1. Explain in a few sentences the way of operation of each of the three detectors.

Canny:

The Canny edge detector uses the fact that the derivative of the Gaussian Laplace can be used to approximate edges in an image. It is typically used after a Gaussian Laplace smoothing on an image. The Canny edge detector uses the sigma value as the input.

Sobel:

The Sobel edge detector is comprised of two 3x3 kernels that are convolved with an image to produce an edge detected graph. This operator requires a threshold as it is not consistent.

Gaussian-Laplace Filtering:

The Gaussian-Laplace Filter is a distance based filter that heavily weights the center cell and decreases in weight outward. This filter approximates the derivative of the image. Once a Gaussian-Laplace filter is applied, region's values reflect it.

2.

The threshold parameter decides how many total “edges” we include. The higher the threshold, the fewer edges we will include. The lower the threshold, the more “edges” we will include. The threshold parameter determines the balance between precision and recall. If we use a higher threshold, we may increase our precision because we will have fewer false positive, however, we

may decrease our recall because some edges may not be greater than the threshold. Conversely, the lower the threshold, the lower our precision since we will likely falsely label edges. But, we will have a higher recall value because we will likely capture most of the “true edges”.

The sigma parameter influences the amount of noise or how smooth the image will be. However, if sigma is too high, we may also miss some edges. If the original image has similar intensities we will likely use a smaller sigma value to get the maximum edge detection.

In the images below, we see that for each image, a different sigma value works best. And, they will each have a different optimal sigma value.

3. Gaussian Laplace Method

Task 3: Edge Detection

1. Explain in a few sentences the way of operation of each of the three detectors.

Canny:

The Canny edge detector uses the fact that the derivative of the Gaussian Laplace can be used to approximate edges in an image. It is typically used after a Gaussian Laplace smoothing on an image. The Canny edge detector uses the sigma value as the input.

Sobel:

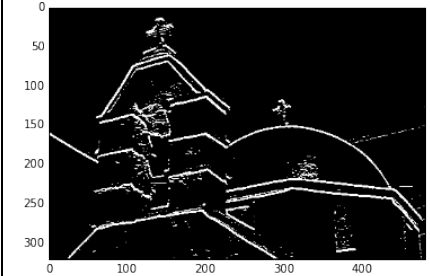
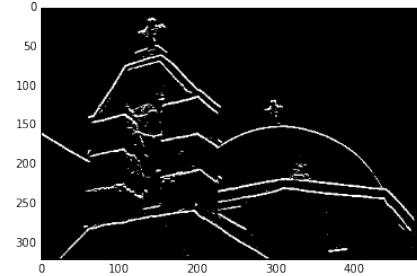
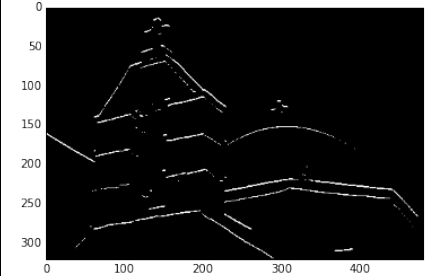
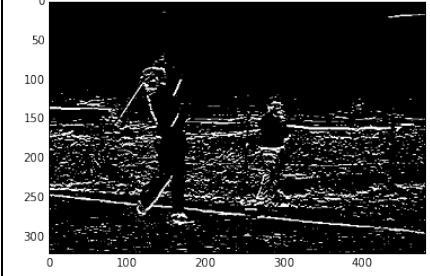
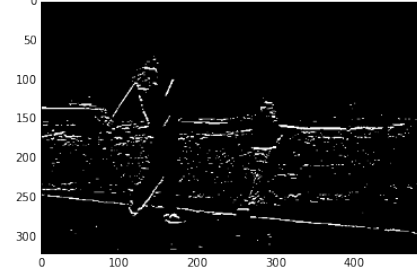
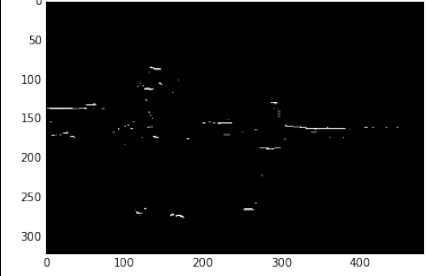
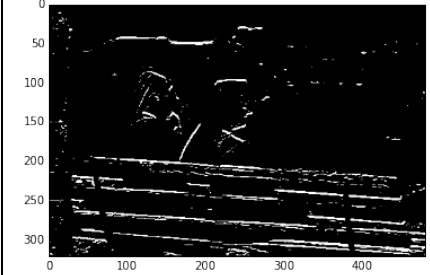
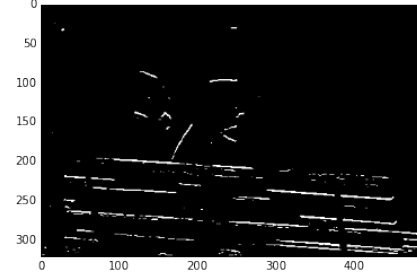
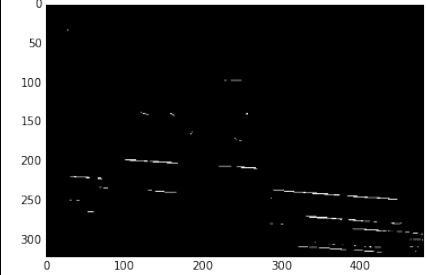
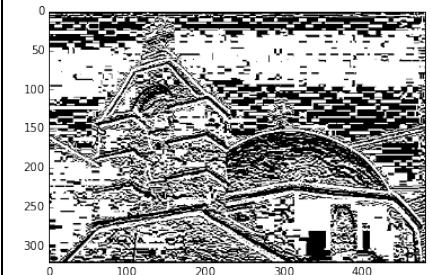
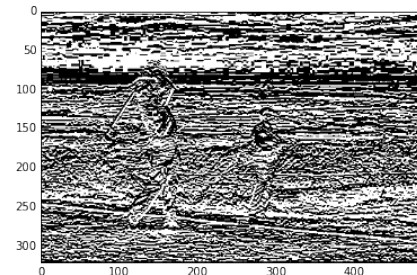

The Sobel edge detector is comprised of two 3x3 kernels that are convolved with an image to produce an edge detected graph. This operator requires a threshold as it is not consistent.

Gaussian-Laplace Filtering:

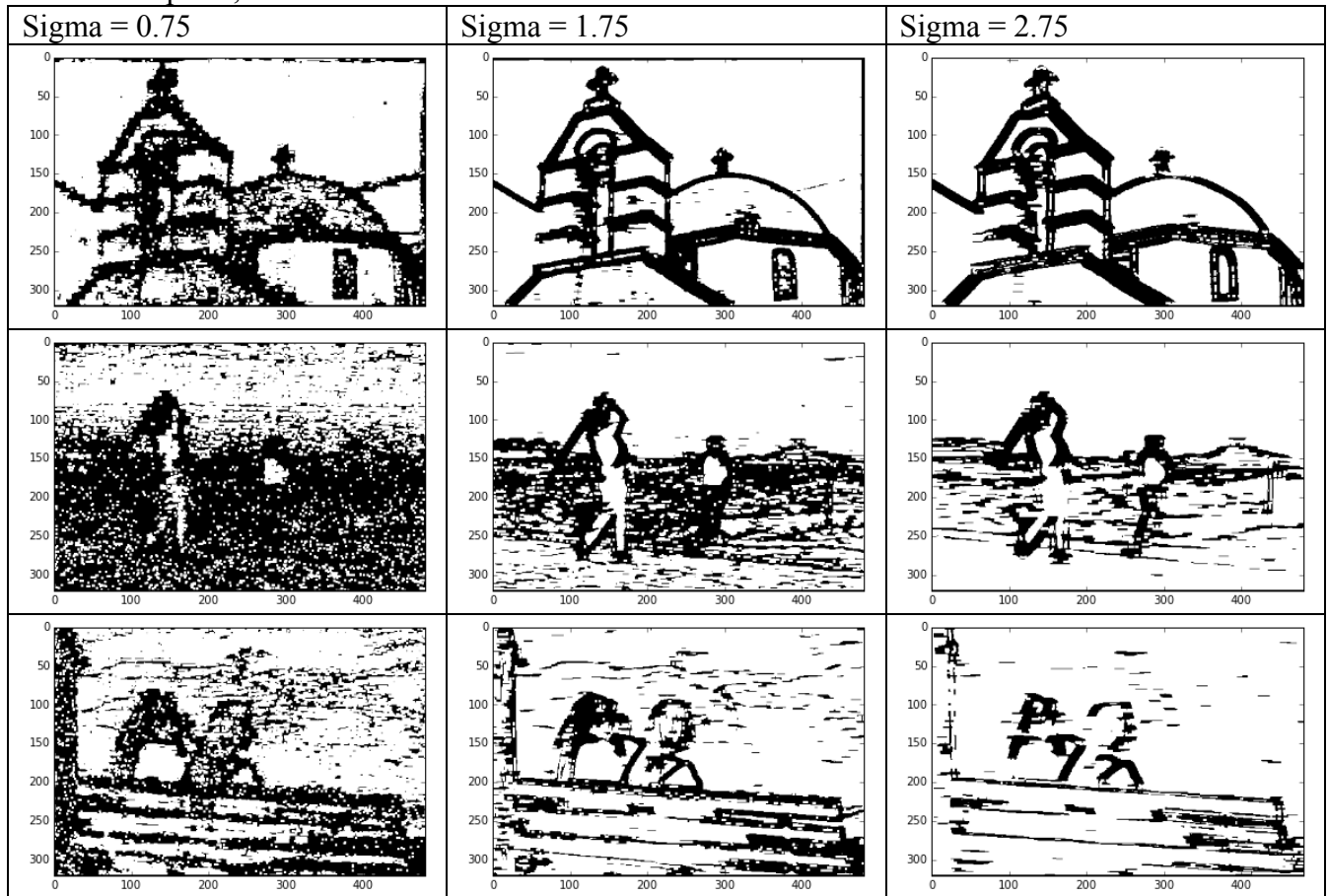
The Gaussian-Laplace Filter is a distance based filter that heavily weights the center cell and decreases in weight outward. This filter approximates the derivative of the image. Once a Gaussian-Laplace filter is applied, region's values reflect it.

2.

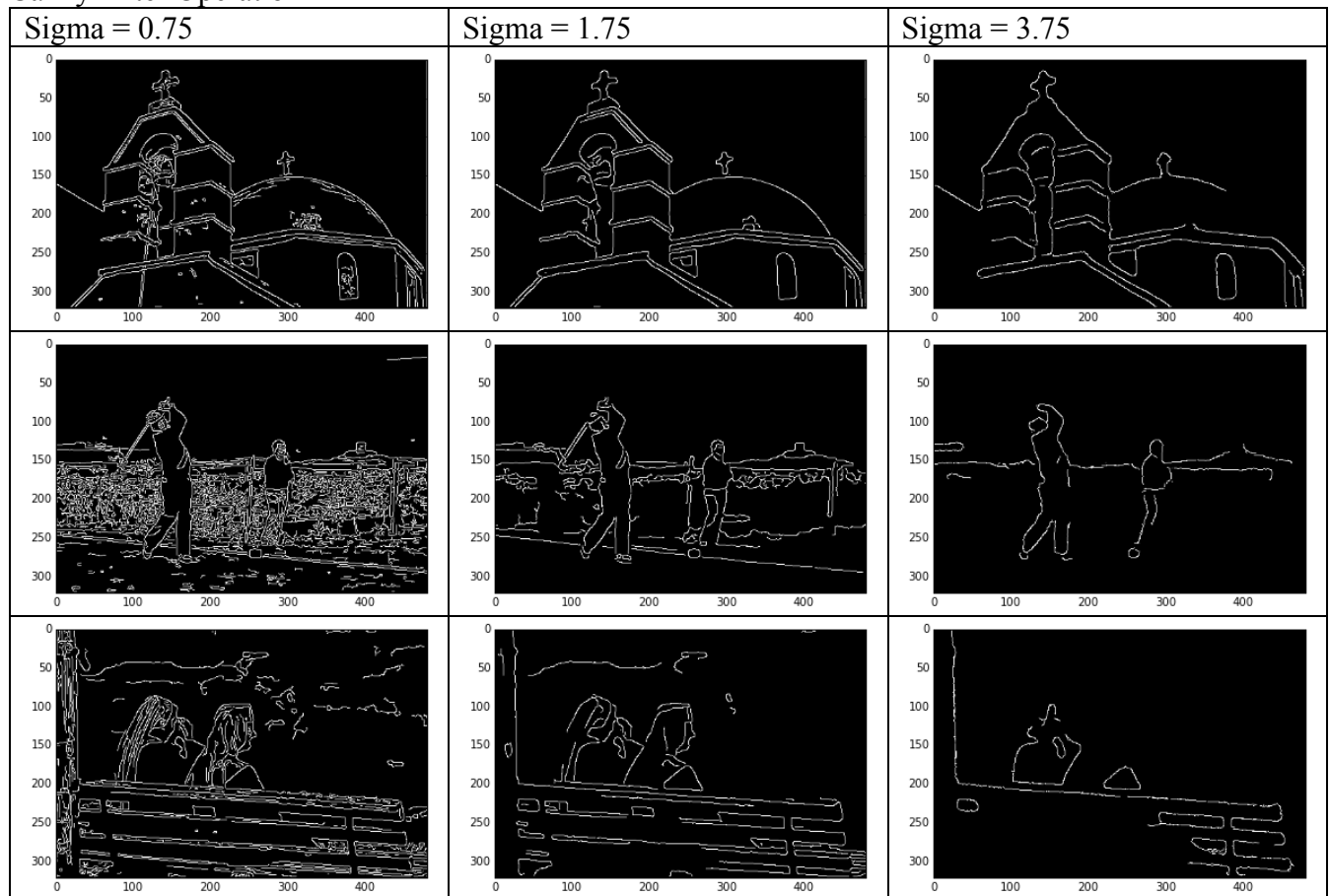
Sobel Operation

Threshold = 0.05	Threshold = 0.1	Threshold = 0.3
		
		
		
Threshold = 0	Threshold = 0	Threshold = 0
		

Gaussian-Laplace, threshold =0



Canny Filter Operation



The threshold parameter decides how many total “edges” we include. The higher the threshold, the fewer edges we will include. The lower the threshold, the more “edges” we will include. The threshold parameter determines the balance between precision and recall. If we use a higher threshold, we may increase our precision because we will have fewer false positive, however, we may decrease our recall because some edges may not be greater than the threshold. Conversely, the lower the threshold, the lower our precision since we will likely falsely label edges. But, we will have a higher recall value because we will likely capture most of the “true edges”.

The sigma parameter influences the amount of noise or how smooth the image will be. However, if sigma is too high, we may also miss some edges. If the original image has similar intensities we will likely use a smaller sigma value to get the maximum edge detection.

In the images below, we see that for each image, a different sigma value works best. And, they will each have a different optimal sigma value.

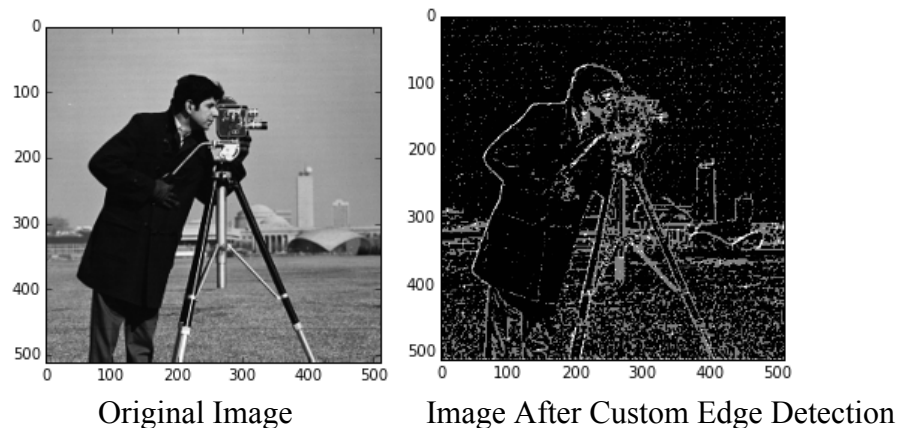
B)

Precision and Recall are important because Precision explains the percentage of edges reported that are truly edges and Recall explains the percentage of edges detected from all the edges. A low recall value means that we are not detecting all of the edges and may mean that we are missing critical edges from the image. This happens when a poor threshold value is used.

A low precision value means we are incorrectly classifying some points: either missing real edges (true positives) or including false edges (false positives). If we're missing real edges, then we are simply not detecting edges. And, if we are including false edges we are adding false edges to the image.

3. Zero Crossing is taken preference over thresholding, and whatever edges zero crossing detects will be in the form of curves.¹

4. Custom Edge Detector



Here are the results of the Custom Edge Detector. This detector first takes the averages of the differences between two horizontal and two vertical pixels to generate a basic edge map. It was able to generate a reasonable image, however it contained lots of noise. A 3x3 min filter was used as a threshold to help remove the noise from the grass. The filter ensured that the point with the lowest value in the 3x3 matrix was within the threshold. Next a global threshold was applied, only accepting pixels above a certain value. The end result is appears to capture all of the major edges. This filter is better than the others because it produces an image with significantly less noise (especially near the grass) than the other filter (Canny or Sobel).

B)

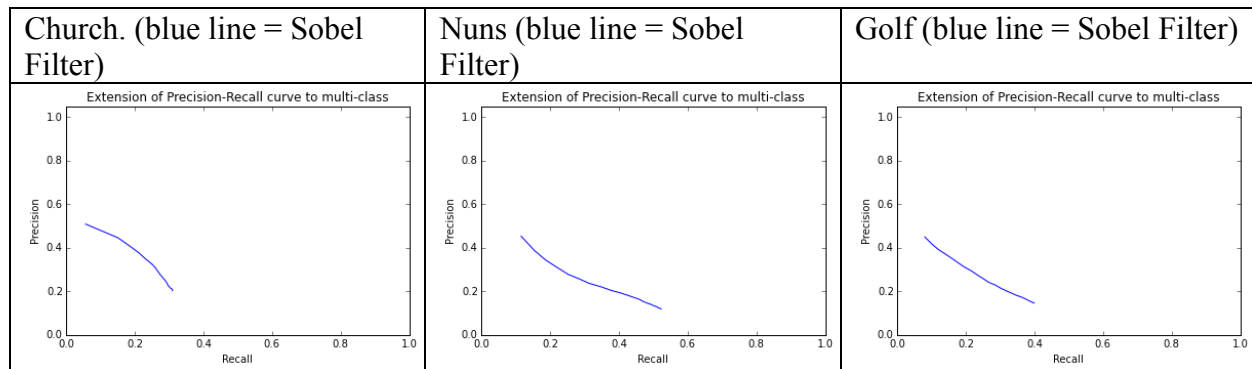
Precision and Recall are important because Precision explains the percentage of edges reported that are truly edges and Recall explains the percentage of edges detected from all the edges. A low recall value means that we are not detecting all of the edges and may mean that we are missing critical edges from the image. This happens when a poor threshold value is used.

A low precision value means we are incorrectly classifying some points: either missing real edges (true positives) or including false edges (false positives). If we're missing real edges, then

¹ <http://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm>

we are simply not detecting edges. And, if we are including false edges we are adding false edges to the image.

Unfortunately, there is a bug in our code and, for some reason the `precision_and_recall_values` is only working for the Sobel method. Sadly, we were unable to debug in time.



3.3b:

We would implement the algorithm as follows, but I didn't suffice for time limitations.

We would compare GT image to the filtered image.

Algorithm:

Traversing the GT image, each time I detect an edge, I would then compare to the 3x3 neighborhood of the filter-image.

If I see an edge in the filtered image,

I increment true positive.

Delete the (first) edge detected.

Otherwise, I would increment false negative (for the undetected edge).

After this for-loop, I've counted True positives and false negatives.

I would count the remaining edges in the filtered images which equals false positives.

I can then derive positive negatives.

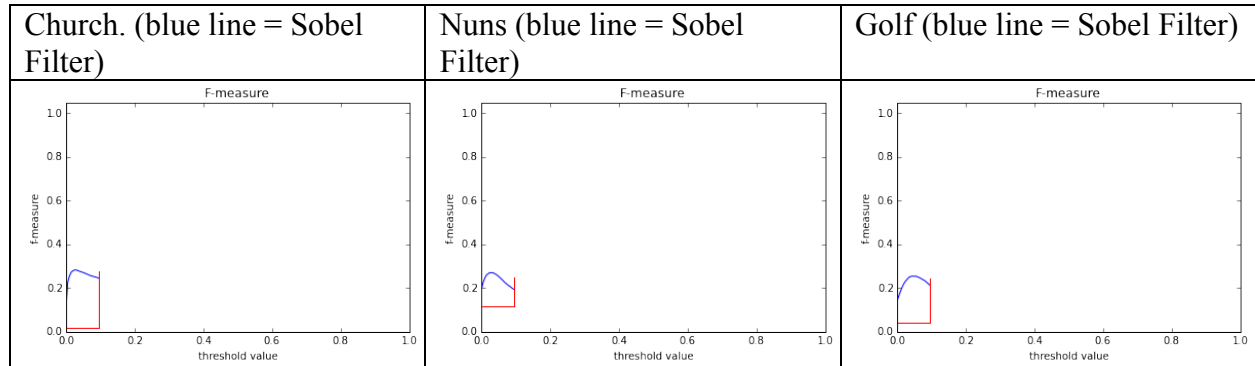
From here, I calculate the precision and recall values and return.

This is more efficient than comparing filtered to GT image.

We're sorry we didn't have time to implement.

3B4.

F-measure charts



Sobel does. But, I am convinced that there is a bug relating to the PR calculation for Gauss-Laplace filter operation.