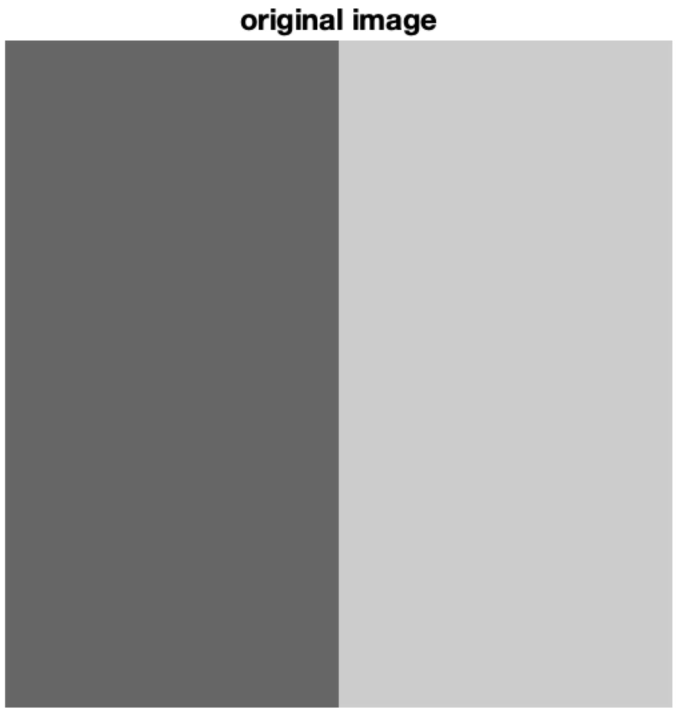
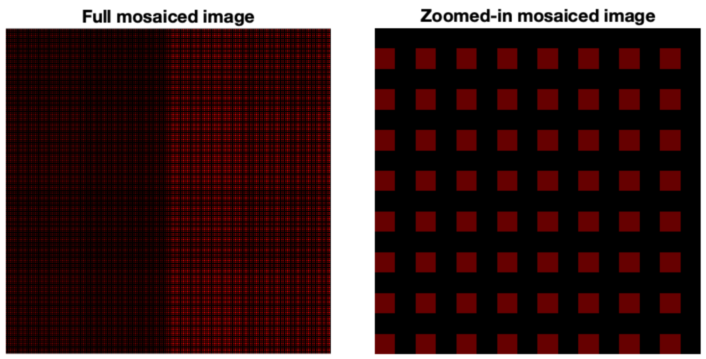
7 October 2020
Fundamentals of Computer Vision

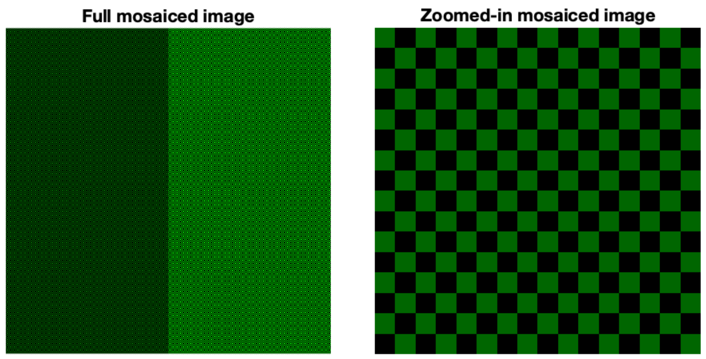
Assignment 1 260807622

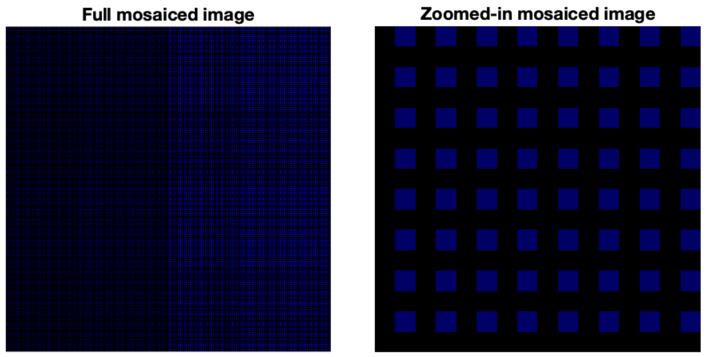
1. **Demosaicing**

   (a) Below is shown the original sharp intensity image that was used along with the full ”mosaiced” images produced and their zoomed-in versions (to show sampling was done correctly).




Full mosaiced image
Zoomed-in mosaiced image


Full mosaiced image
Zoomed-in mosaiced image



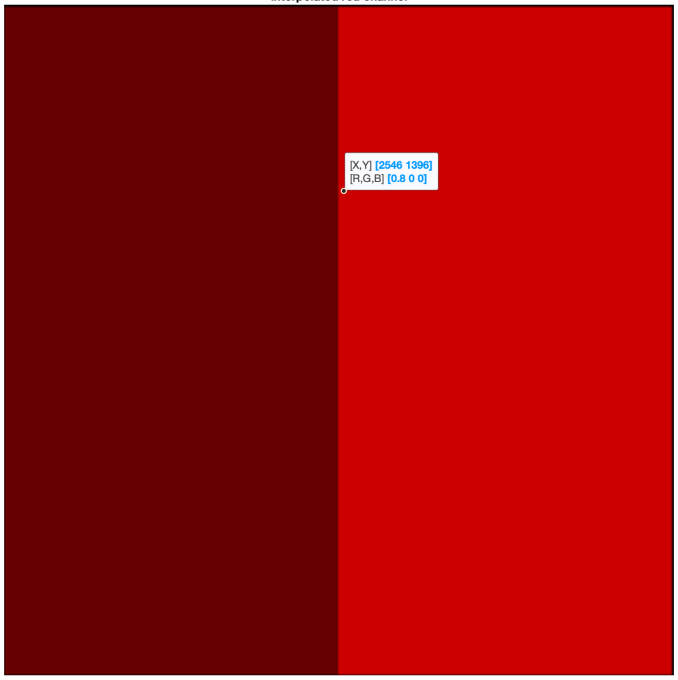
(b) For this part the filters used were:
For both the red and the blue filters:

$$\frac{1}{4}\begin{bmatrix}0 & 1 & 0\\ 1 & 4 & 1\\ 0 & 1 & 0\end{bmatrix}$$

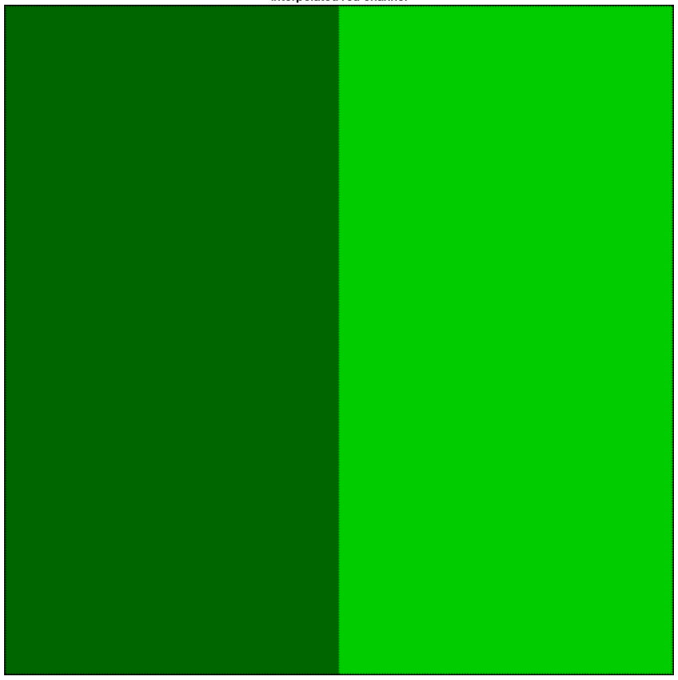
For the green filter:

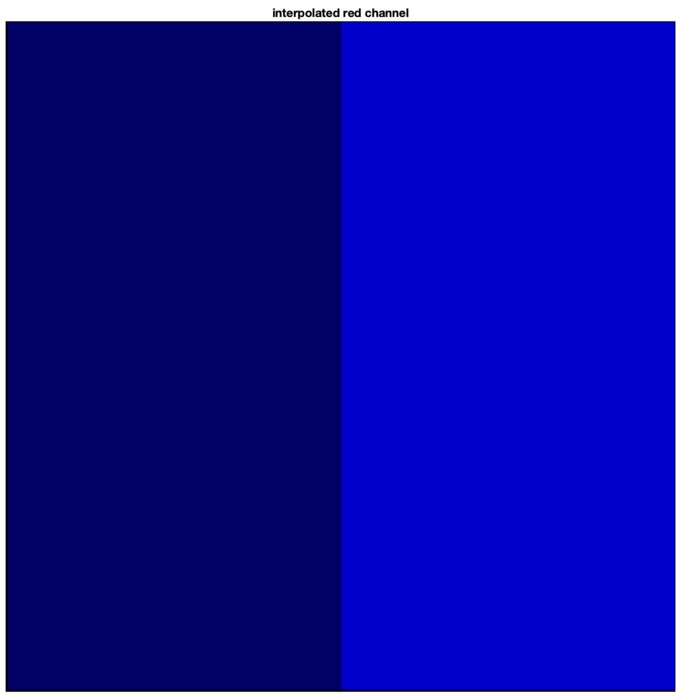
$$\frac{1}{4}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Below are the interpolated images of each channel interpolated red channel



interpolated red channel

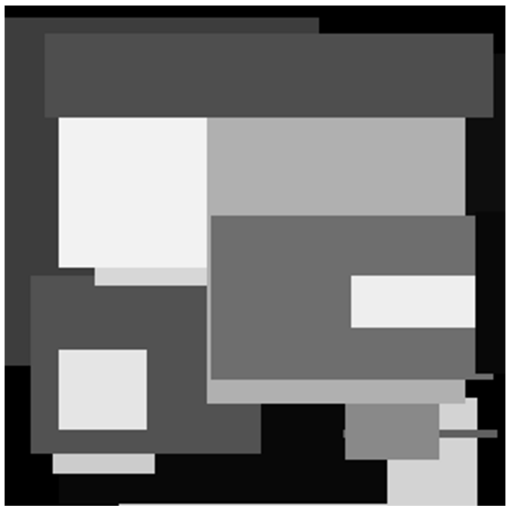




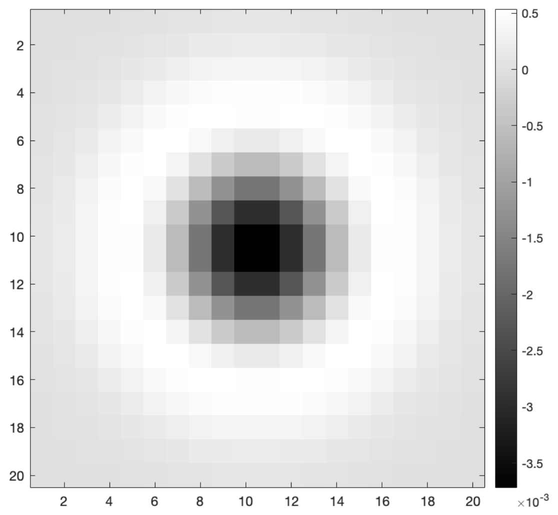
(c) We see for the interpolated images that the value of the pixels on one side of the edge are *almost* equal to any other pixel residing on the same side of the edge. That is because we interpolated in taking averages of neighboring pixels of the corresponding color, which, because of the way we constructed the image, all had equal value.
However, if we get close to the edge, we will find that the pixel values will vary in getting closer and closer to 0.5 as we move towards the edge.

2. **Edge Detection**

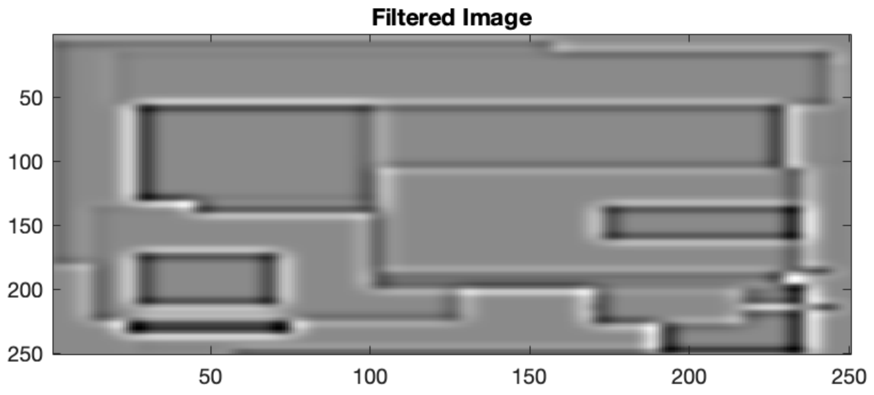
(a) Intensity image consisting of 100 rectangles of different positions, sizes, and grey levels (all generated randomly):

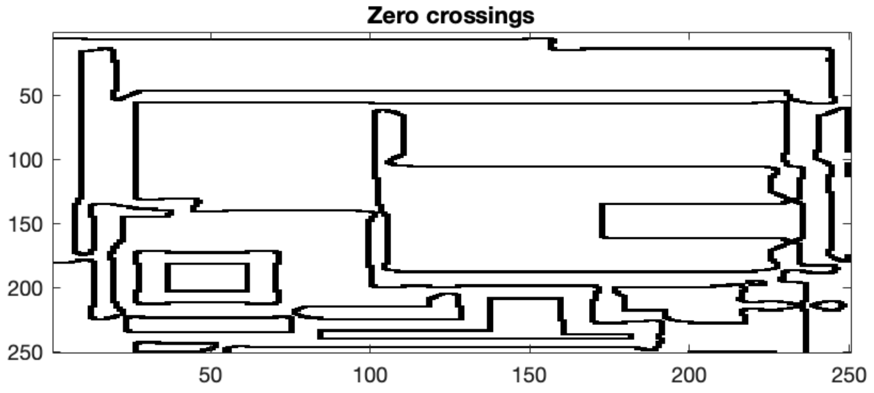


(b) Laplacian of a Gaussian with size = 20 and $\sigma = 3$.

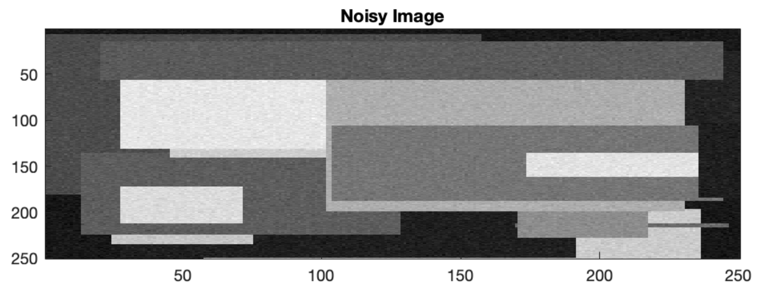


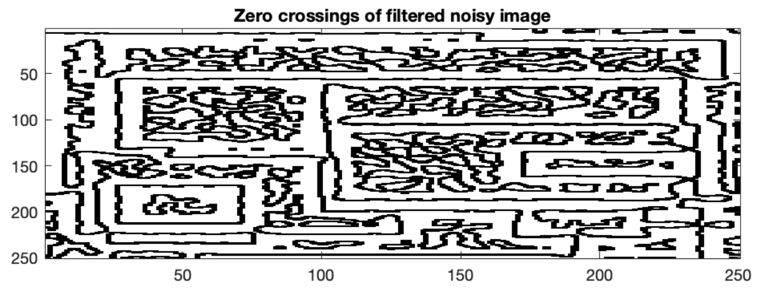
(c) Filtered image with the LoG filter above and zero crossings found:



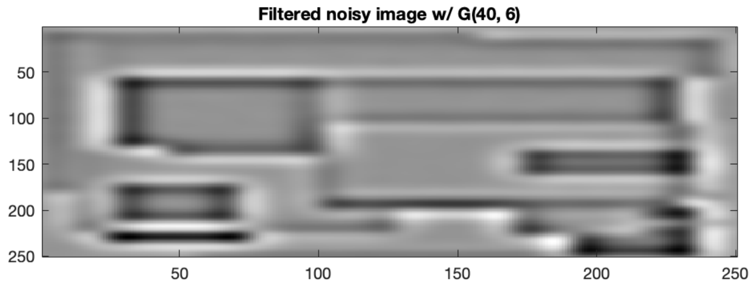


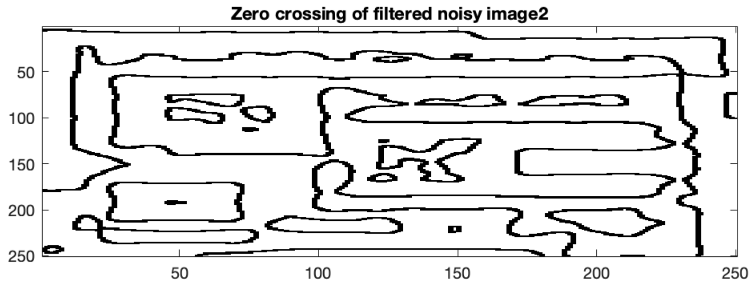
(d) In this part we added noise to the image before filtering it again with the LoG filter and then finding the zero crossings once more, so below are shown the noisy image, filter noisy image, and zero crossings found:





(e) Now we change our LoG filter by making its size 40 and $\sigma = 6$. Below we show the noisy image filtered by this new filter and the zero crossings that were then found.

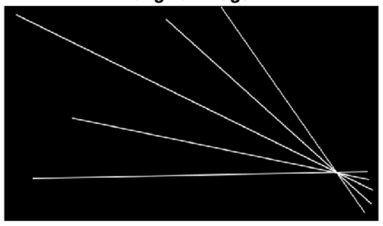


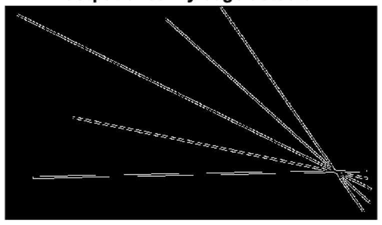


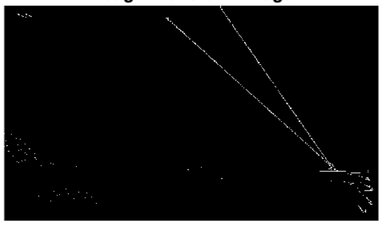
(f) Comparing the bottom images from parts (d) and (e), we can see that in applying a LoG filter of larger size and larger standard deviation, we identify less of the noise as edges. Our zero crossings from the filter with larger size and σ are closer to those we found for the image without noise but with a LoG of size 20 and $\sigma 3$.

3. **Vanishing Point Detection using Hough Transform**

   (a) Below we show the original image, the output of the canny edge detector from `makeEdgeList.m`, and the Hough transform image. The Hough transform algorithm was not successful in voting for a good number of points but it seemed to have identified the obvious hough peak properly.


original image


Output of canny edge detector


Hough Transform image

   (b) For the implementation used in part (a), we use nested for loops. The outer loop loops through the number of edges, call that number E. In the inner loop there is an if branch, in which both branches iterate over the number of rows, or columns of the image, respectively. In this case, we assume $numrows = numcolumns = N$, such that the image is a $N \times N$ pixel image. This gives us an inner loop with

N iterations and an outer loop with E iterations. Thus the time complexity of the algorithm is

$$\mathcal{O}(EN) \tag{1}$$

Now the implementation in which we check each pair of edges for intersection, we still have nested for loops and the outer loops still iterates E times. The difference is in the inner loop. The number of iteration goes as follows:
In the first outer loop iteration, the inner loop check every possible pair of edges which contains the first edge, that is, (E - 1) iterations.
In the second outer loop iteration, it checks for every pair of edges which contains the second edge, except one of them has already been checked in the first iteration, so that is (E - 2) iterations.
Similarly for the third outer loop iteration, the inner loop will have E - 3 iterations up until the outer loop will check the possible pairs for the last, i.e. 1 iteration.
So in total the inner loop will run (E-1)! times.
Thus the cime complexity of this algorithm is

$$\mathcal{O}(E(E-1)!) = \mathcal{O}(E!) \tag{2}$$

So the first algorithm is faster