CS543 – Assignment 2

Pallaw Kumar (pallawk2)

Question 1.

1. Method Description

Step 1 - To remove artifacts added the argument boundary='symm' to the convolve2d function.

I discovered some artifacts at the picture boundaries after running the raw code. I believe the artifacts are produced when the convolution is applied up till the boundary. As a result, I added the argument boundary='symm' to the convolve2d function. The boundary="symm" argument specifies that symmetric boundary conditions should be used, which means that the image is mirrored at the boundary before applying convolution. This can help avoid artifacts that can arise from applying the filter to pixels with different numbers of neighbors.

Step 2 - Applied Gaussian smoothing to the input image to remove noise and make the edges more prominent.

To get more reliable estimates of the gradient in this section, I utilize derivatives of Gaussian filt ers. We can calculate the parameter truncate using sigma and w, which is the width of the Gaussi an kernel.

Step 3 - Applied non-maximum suppression to thick the edges and obtained an edge map.

I understood that thin margins were being produced via non-maximum suppression. I next determine the gradient's orientation using dx and dy. Then, I locate two points on and across from the gradient's direction, and I apply the interpolation. Finally, the point will be set to 0 if it is not the maximum.

The best model was found using sigma σ=20 and a window size of 7 (after smoothing and non-maximum suppression operation).

2. Precision Recall Plot

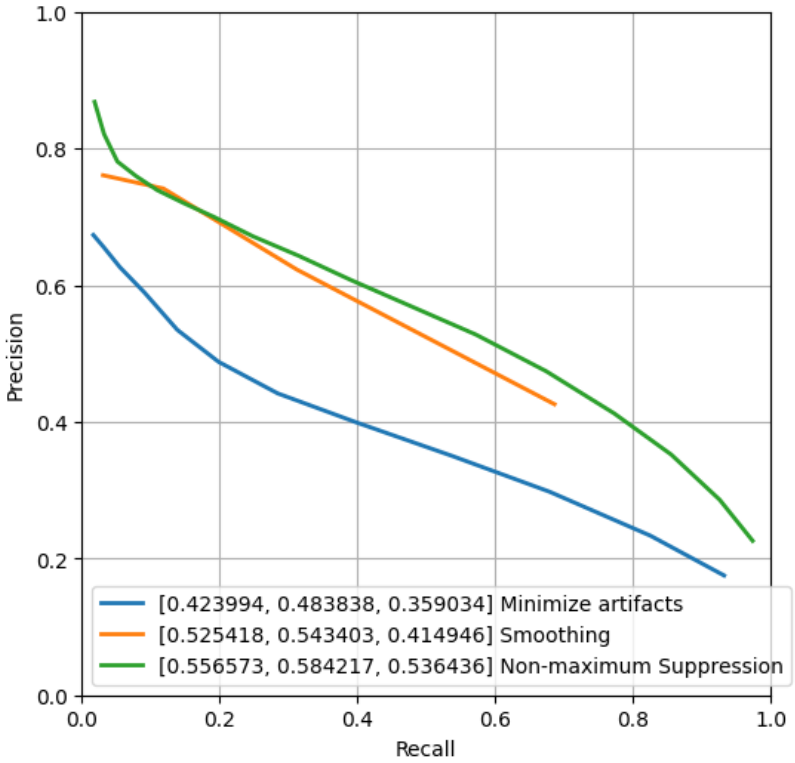


Fig. 1 – Precision Recall Plot of contour detection

3. Results Table

Method	Overall max F-score	average max F-score	AP	Runtime (seconds)
Initial Implementation	0.404	0.460	0.315	0.006
Warm-up [remove boundary artifacts]	0.4239	0.4838	0.3590	0.015
Smoothing	0.5093	0.5093	0.3197	0.0213
Non-maximum Suppression	0.5564	0.5837	0.5364	0.9877
Val set numbers of best model [From gradescope]	0.5565	0.5837	0.5364	12.42s/it

4. Visualizations

Minimizing the artifacts

Factors	Before	After
Overall max F1 Score	0.4040	0.4239
Average max F1 score	0.4589	0.4838
AP	0.3153	0.3590
Runtime (in seconds)	0.0167	0.0157

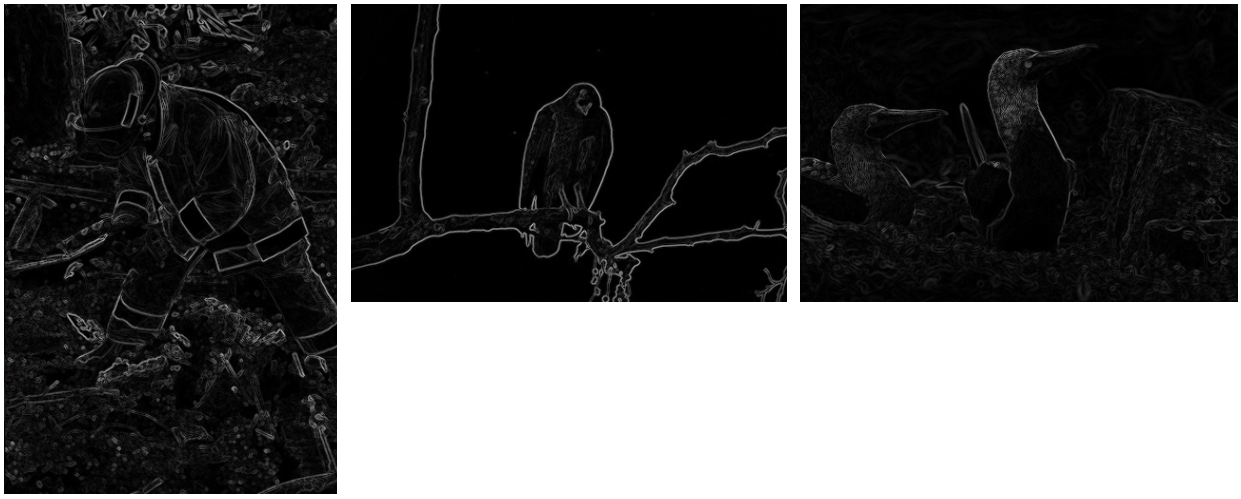


Fig. 2 – Key observation - the edge artifacts are no longer visible but there is not much difference from demo code in terms of visualization.

Smoothing

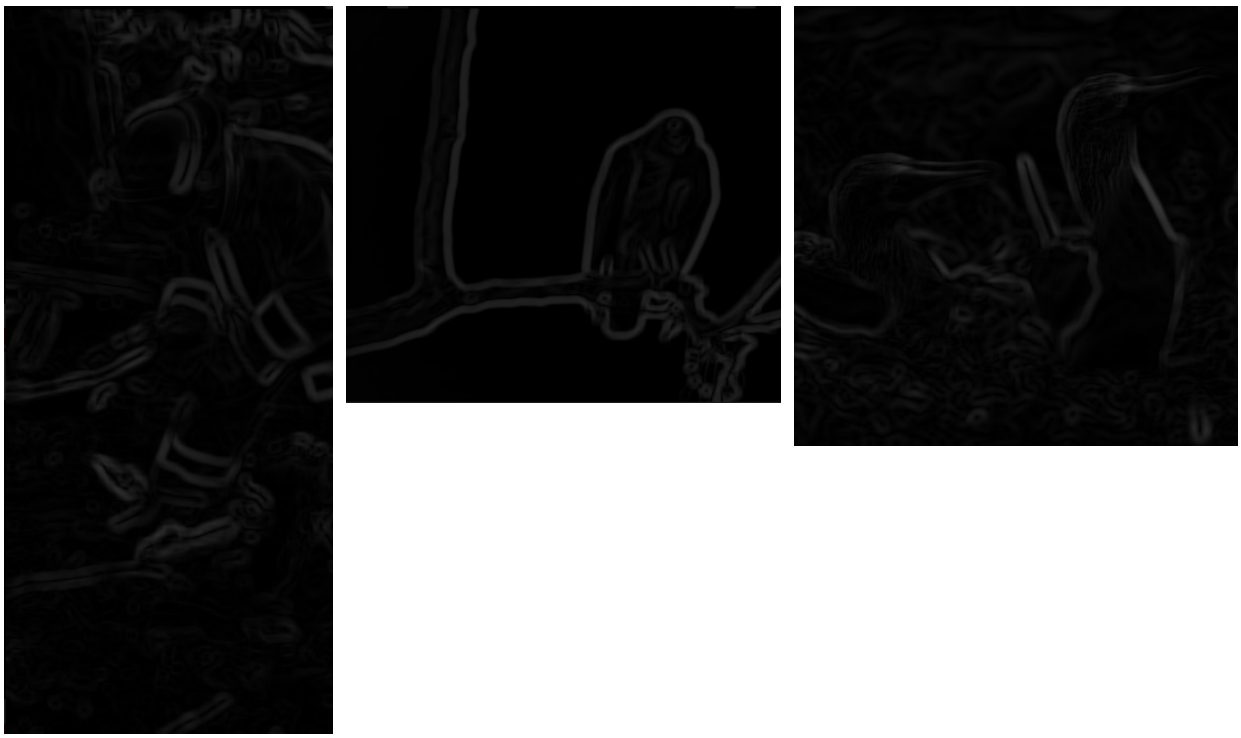
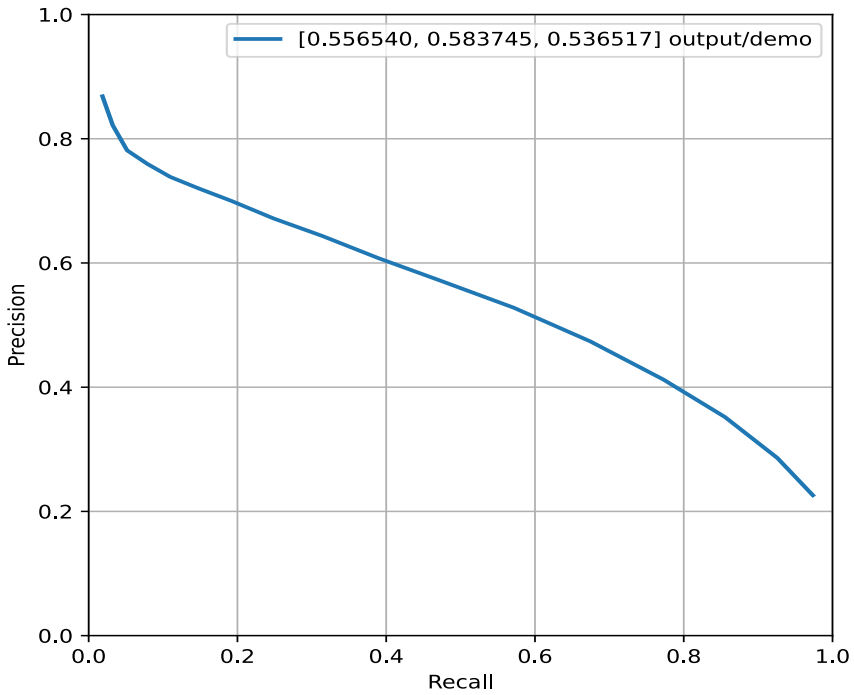


Fig. 3 – Key observation - the edges are thicker and there is less noise. The edges became thicker and more viscous when the gaussian filter was applied, yet they were impossible to differentiate from one another when close to other edges. For example, face of the person in the first image cannot be distinguished. Lines on the cloth of person are thicker and more blurred.

Factors	Before	After
Overall max F1 Score	0.4239	0.5093
Average max F1 score	0.4838	0.5695
AP	0.3590	0.3197
Runtime (in seconds)	0.0157	0.0213

Non-maximum Suppression



Precision Recall graph of contour detection after Non-maximum suppression

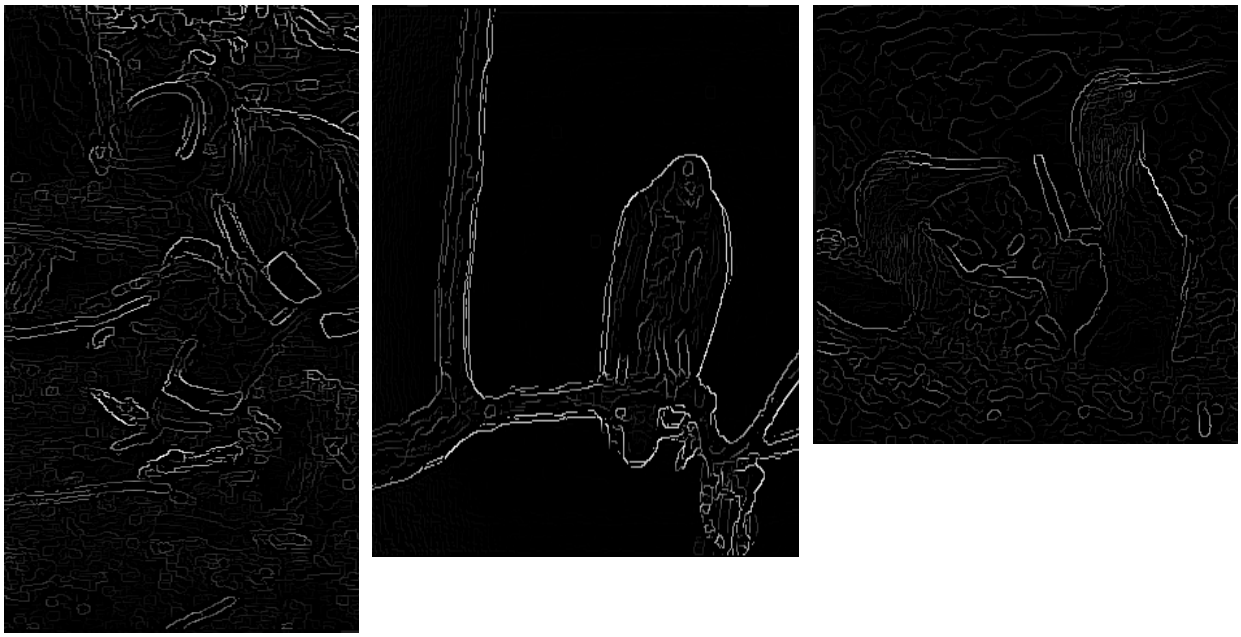


Fig. 4. – Key observation - Compared to the other ways, the lines/edges are extremely fine, and the details are easier to see. For example, lines on the cloth of person in the first image has become thinner and even so, certain edges are invisible. For example, it is impossible to see or recognize the man's face and helmet in the first image. Similarly in the third image it is difficult

to identify the middle bird. weak edges have been removed and strong edges have been retained in the edge map.

5. Bells and Whistles

Method	Overall max F-score	Average max F-score	AP	Runtime (Seconds)
Best base Implementation	0.5564	0.5837	0.5364	0.9877
Bells and whistle (1) [extra credit])	0.5542	0.5846	0.5375	1.1481
Bells and whistle (2) [extra credit])	0.5401	0.5675	0.5222	1.1145
Bells and whistle (3) [extra credit])				

Question 2

1. Method Description

Step 1 - Computed the horizontal and vertical gradients of the image using a filters [-1,0,1] and transpose of [-1,0,1].
Step 2 - Computed the structure tensor at each pixel in the image, which is a 2x2 matrix that encodes the second moments of the image gradients. The structure tensor is defined as:
M = sum(w(x,y) * [Ix^2(x,y), Ix(x,y)*Iy(x,y); Ix(x,y)*Iy(x,y), Iy^2(x,y)])

where Ix and Iy are the horizontal and vertical gradients of the image, and w(x,y) is a Gaussian window function centered at (x,y). Performance tests were conducted on the following sigmas: [0.4,0.8,1.5].
Step 3 - Computed the Harris response at each pixel in the image using the following formula:

$$R = \det(M) - k * (\text{trace}(M))^2$$

where det(M) is the determinant of the structure tensor, trace(M) is the trace of the structure tensor, and k is an empirical constant set to a small value. The *k* was tuned. The following values were tested: 0.04,0.05.

Step 4 – Applied thresholding in the Harris response image to obtain a binary corner map. Pixels with a high Harris response are considered corners.

- **The best model was discovered with the following hyperparameters: k=0.04 and sigma=0.5, without non-maximum suppression.**

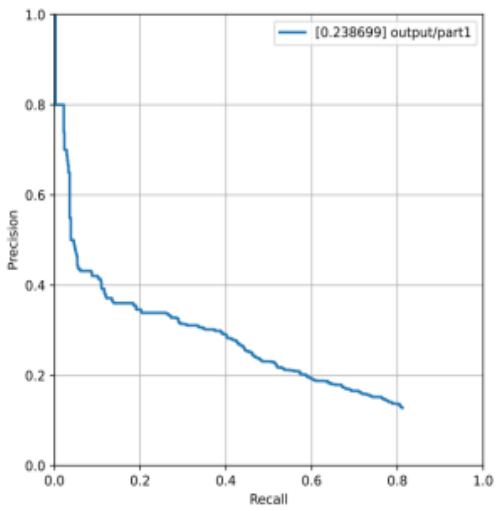
Step 5 - Performed non-maximum suppression on the corner map to eliminate weak corners that are close to strong corners. For non-maximum suppression, the window size was tuned. Window

sizes 3 and 7 were tested. Each pixel was altered to 0 if the reaction of its pixel neighbors was higher than its own. According to the experimental findings, performance is negatively impacted by huge window sizes.

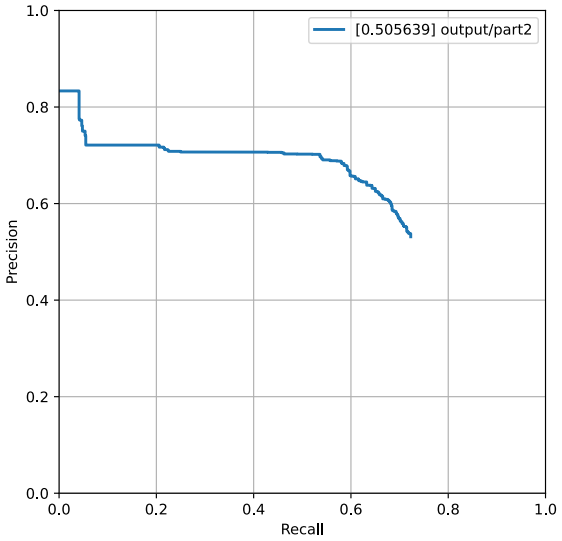
- **The best model with nonmaximum suppression has the following hyperparameter values: k=0.04, sigma=0.8, and W = 3.**

2. Precision Recall Plot

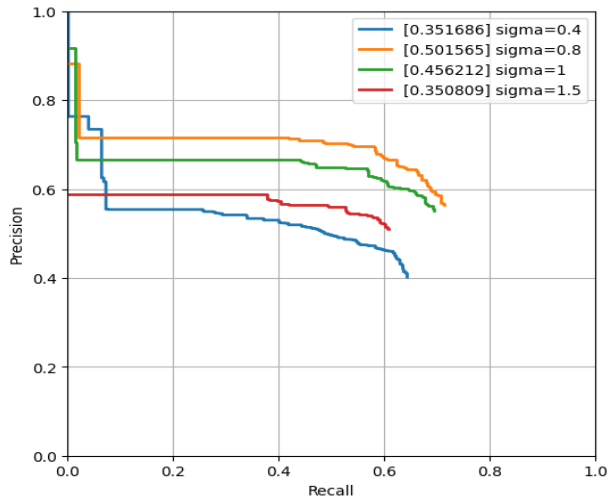
Without NMS



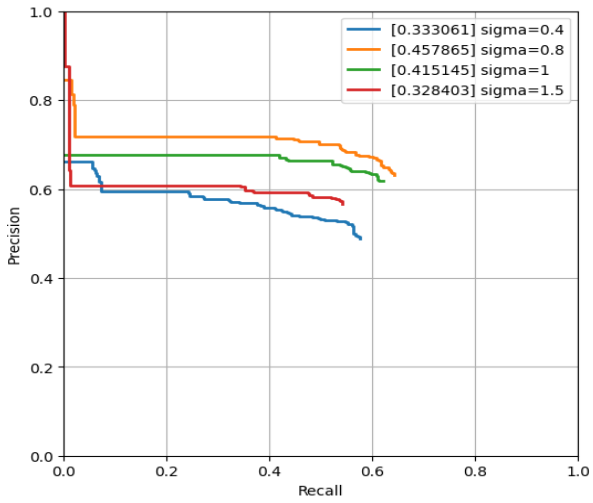
With NMS



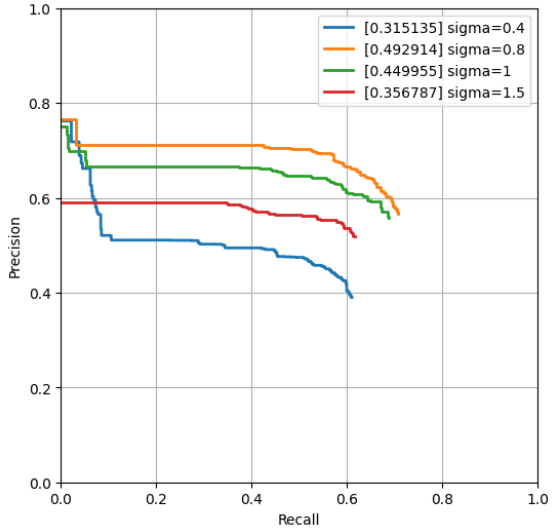
Window Size = 3 and k = 0.04



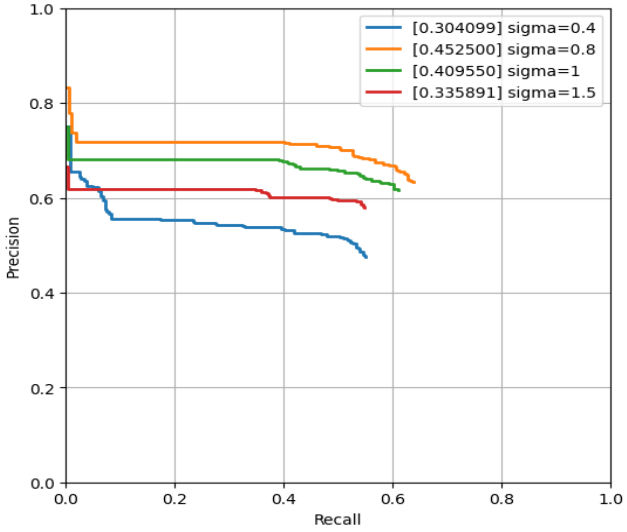
Window Size = 7 and k = 0.04



Window Size = 3 and k = 0.05



Window Size = 7 and k = 0.05

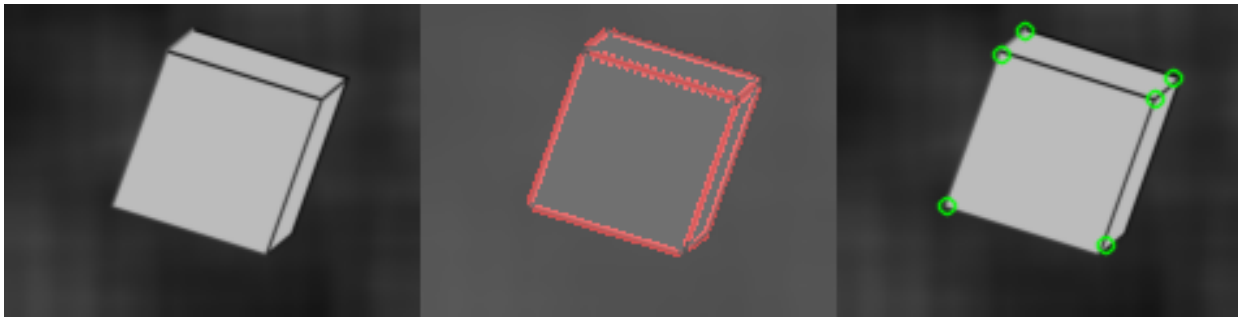


3. Results Table

Method	Average Precision	Runtime (seconds)

Random	0.002	0.0010
Harris W/o NMS	0.2387	0.0046
Harris W/ NMS	0.5056	0.4903
Hyper-parameters tried (K=0.04, σ =0.8, W=3 with NMS)	0.5015	0.4961
Hyper-parameters tried (K=0.05, σ =0.8, W=3)	0.4929	0.5001
Val set numbers of best model [From gradescope]	0.5056	791.83it/s

4. Visualizations



The cube's corners can be found with the corner detector. Yet, it misses the tucked-away corner in the back. It's possible that the black edge blends into the background, making it challenging to see gradient changes.

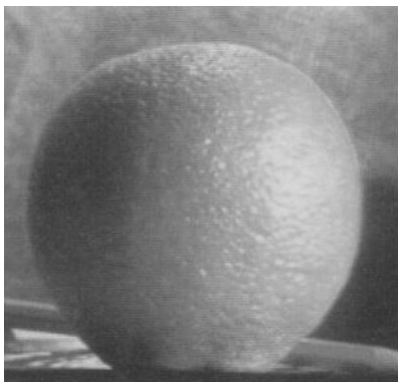
It is unable to recognize the corners of leaves and the walls of windows or curtain panels. The corner detector is capable of picking up on a variety of building façade corners, as well as pedestrian corners and corners of leaves. For building it is not able to detect all the corners, it's possible that the space between pixels of building is insufficient and that many of them are suppressed because of the window size used for non-maximum suppression. Also, illumination and poor-quality image quality may has affected the corner detection.

The corner detector performs a decent job at detecting the corners of the façade, corners on walls, and trees. The sidewalks and windows in the right side of the building still cannot be distinguished as details (perhaps because of the sills' orientation or projection, which may have impacted the computation of the gradient). Resolution of the building in the photograph (e.g., the size impacting the gradient of pixels) likely prevents the detection of the corners of the building buried to the left. It is working doing better than in the Beckman image.

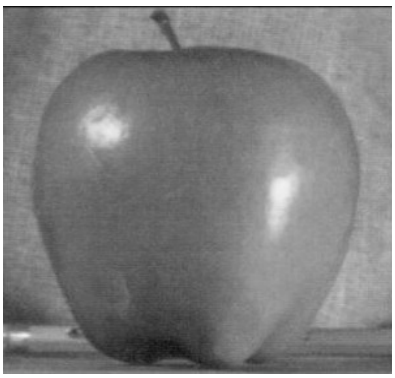
Question 3

1. Method Description

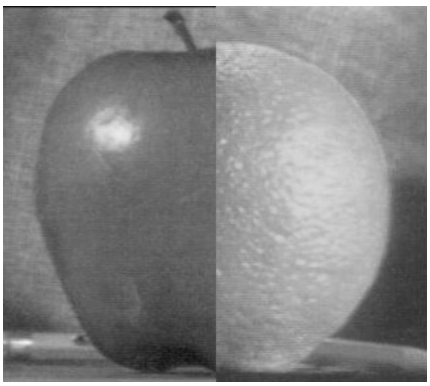
The Laplacian stack for each image was constructed by stacking the gaussian differences from the gaussian stack. The "difference of Gaussians" function from the skimage library was used to create the difference of Gaussians. We pass the sigma of the gaussian corresponding to each level on the original image for each image in the stack. The "gaussian" function from the skimage library was used to construct a gaussian stack for the mask. The number of stack tiers and various gaussian sigmas were tested. Blended the Laplacian pyramids by taking a weighted average of the corresponding levels in each stack. The weights are determined by a mask image that specifies the regions of each image to be blended. The blended image was reconstructed the by adding up the levels of the blended Laplacian stack. The blending's visualization performance improves as the levels and sigma are increased. eight stack were used to build the final mixing.



Orange



Apple



Oraple

Implementation using the vertical mask

Blends of your choice

A function was developed that produced a mask from reference coordinates in x and y. Instead of a mask, the blending function now expects an array of coordinates in the picture coordinate system. The coordinates are used by the mask function to interpolate the values in the direction of the mask. The interpolation makes use of the "interpolate function from Scipy library" and the fixed picture dimensions. Thus, the photos were resized. And finally, both the images and mask image were passed to the blend function written for the last problem.

Image 1

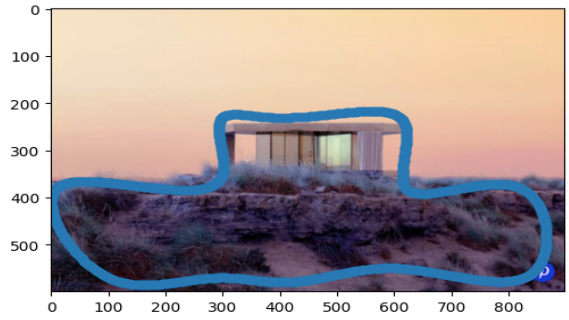
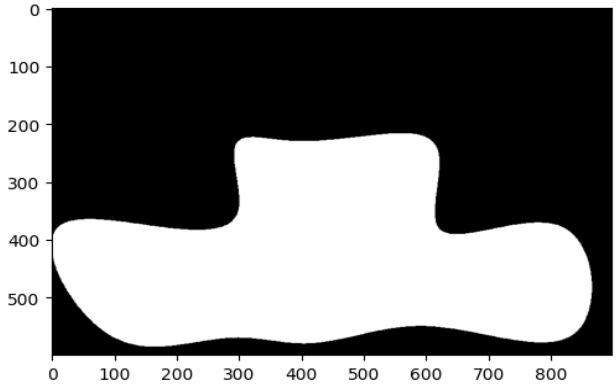
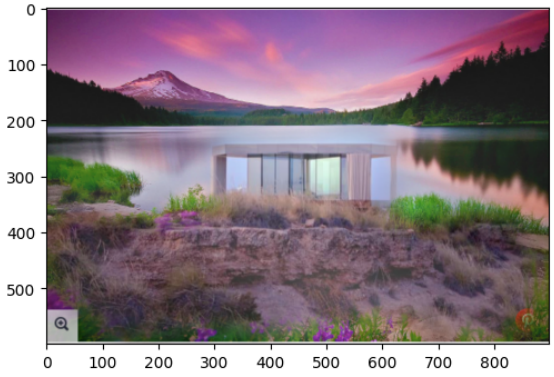


Image 2



Mask Image



Blended Image