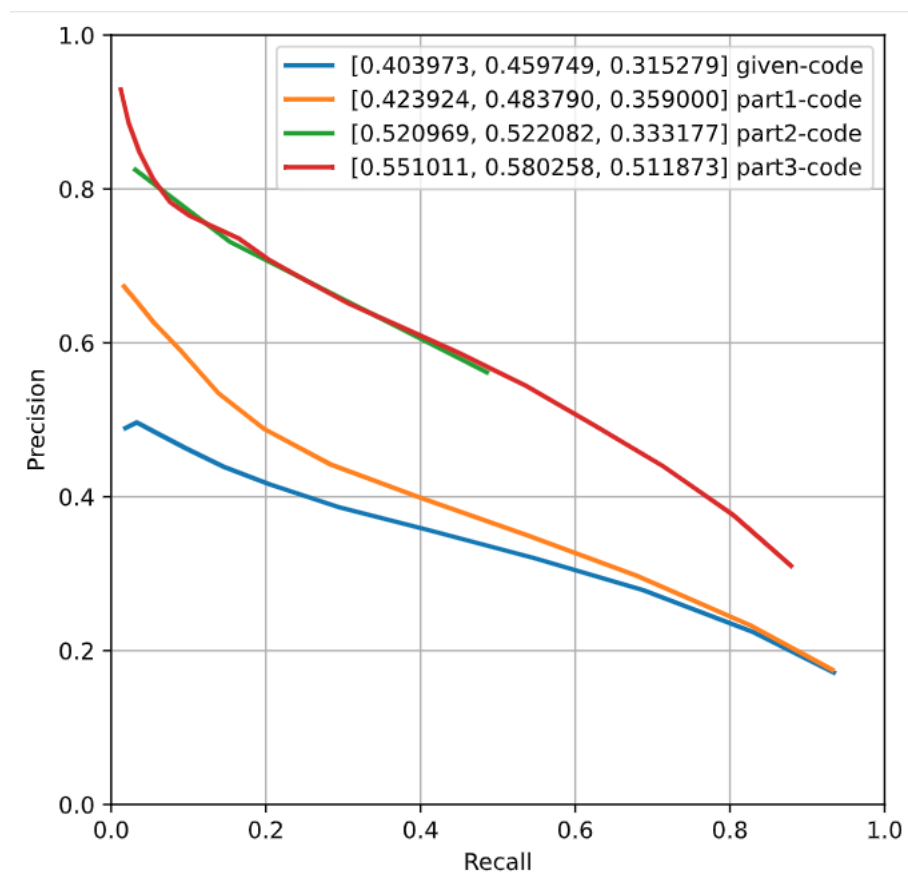


Contour Detection - Solution

Method Description:

For Part 1(Warm-up), to reduce the image artifacts, I tried padding the image on its left, right, up, down directions 3 pixels, which fits the size of the filter. The convolution will have larger overlap area with the images, which will reduce the artifacts of the image boundaries. Then I used the gaussian filter from scipy.ndimage to achieve the gaussian filter, start from 1, I tried sigma = 2, sigma = 3. From 1 to 2, the performance increased, and from 2 to 3, the performance decreased. Eventually after multiple experiments I picked sigma = 2.5 as the best parameter. For the non-maximum suppression, I first calculate the angle (in radians) between the positive x-axis and the point (x, y) in the xy-plane with `np.arctan2()`. Then I continue to determine the direction of the edge at each pixel based on its gradient direction angle. This is done by comparing the gradient angle to a set of predefined threshold values, which correspond to four different edge directions: horizontal, vertical, and two diagonal directions, in other words, I round the gradient direction angle to $45 \cdot n$, n is the integer. However, even though I tried multiple parameters, the eventual result still have 0.001 distance to the final results.

Precision Recall Plot:

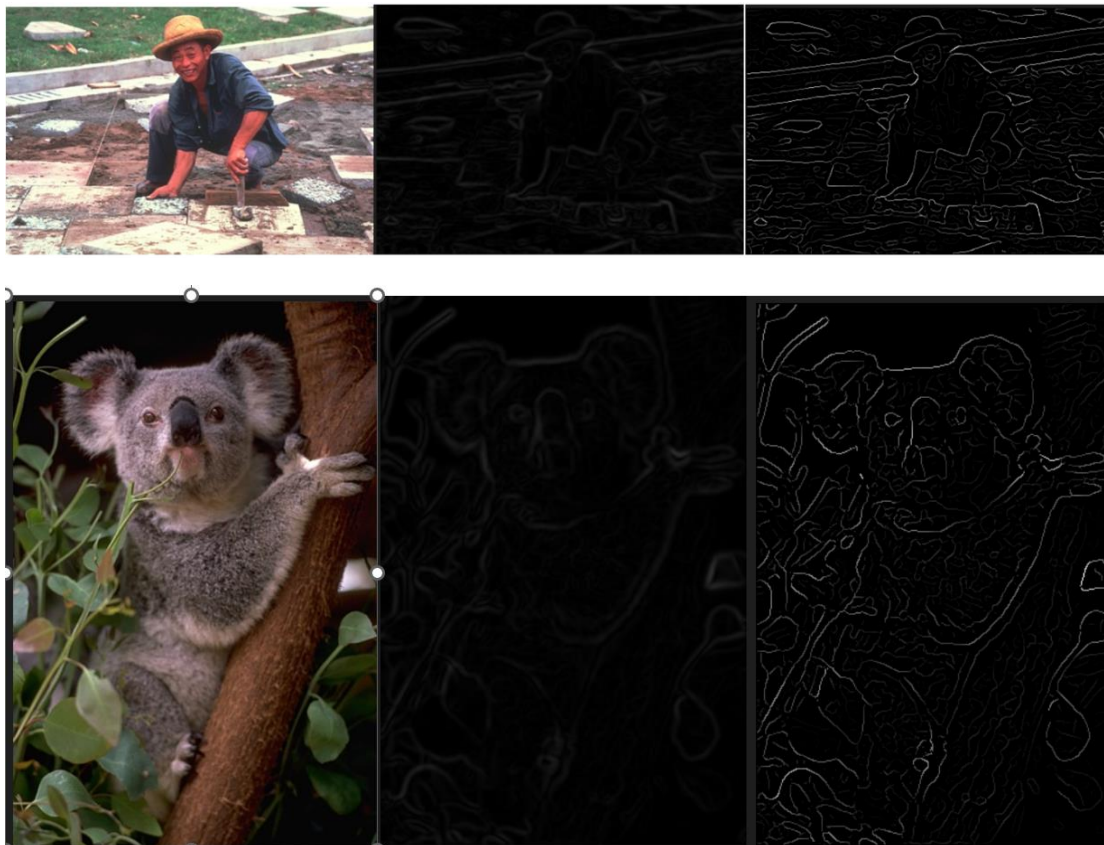


Results Table:

Method	Overall	F-	Average max	AP	Runtime(seconds)
--------	---------	----	-------------	----	------------------

	max score	F-score		
Initial implementation	0.404	0.460	0.315	0.006
Warm-up [remove boundary artifacts]	0.424	0.483	0.359	0.006
Smoothing	0.520	0.522	0.333	0.007
Non-maximum Suppression	0.551	0.580	0.511	0.147
Val set numbers of best model [From gradescope]	0.551	0.580	0.511	0.147

Visualizations:





My code works relatively well for the contours that was formed by large piece geometry pieces. However, when the image is shattered, or having condensed geometries, or especially with condense, line-like objects(like grass), my code does not behave so well. Also, my contour detected is not smooth enough and I also did not completely remove the artifacts. I think it might be the problem of implementation of NMS. Or the sigma of Gaussian filter.

Corner Detection - Solution

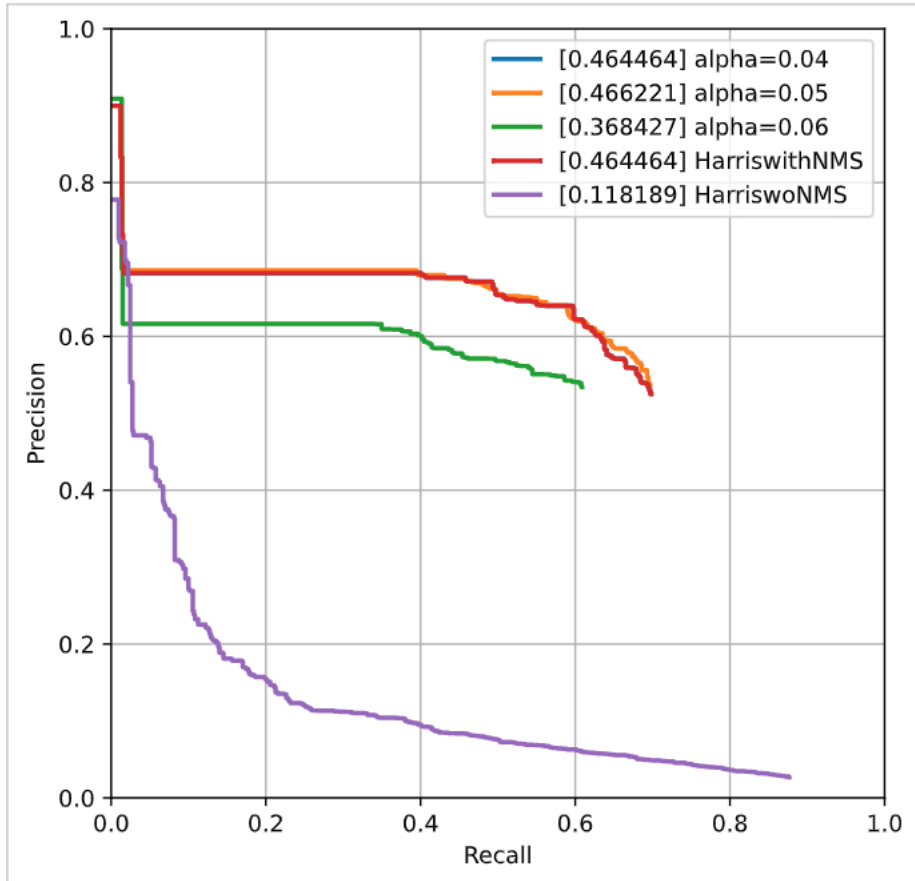
Method Description:

In my implementation of the Harris Corner Detection, I first calculated the image gradients (or partial gradients of each pixel) with the first-order derivative filters. Then I calculated the moment matrix (or structural tensor) with the calculated image gradients.

$$M = \begin{bmatrix} \sum_{x,y} w(x,y) I_x^2 & \sum_{x,y} w(x,y) I_x I_y \\ \sum_{x,y} w(x,y) I_x I_y & \sum_{x,y} w(x,y) I_y^2 \end{bmatrix}$$

Based on the moment matrix, I was able to calculate the response with $\det(M) - k * (\text{trace}(M))^2$. For each pixel are thresholded to find the locations of potential corners. However, this can lead to multiple detections of the same corner due to noise or overlapping regions. So I applied a NMS to the response to sidestep that problem. To reach the maximum performance, I adjusted parameter multiple times, I tried window size from 3-6, and alpha from 0.04-0.06, eventually found that when $\alpha = 0.05$, window size = $3*3$. The AP reaches the highest.

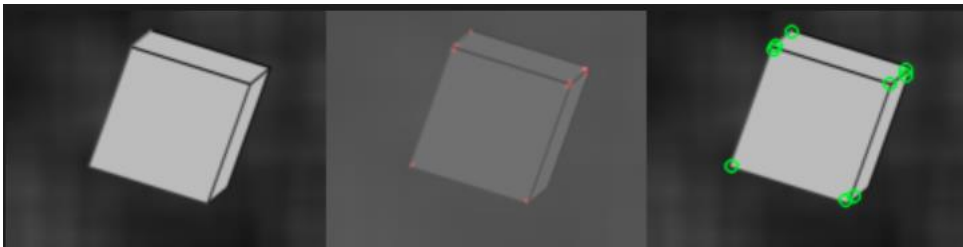
Precision Recall Plot:

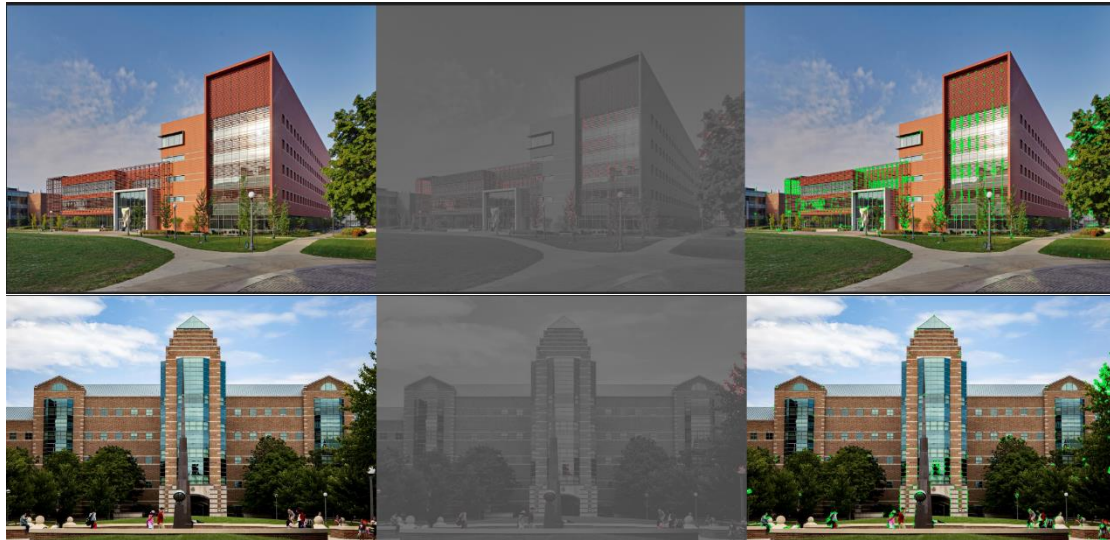


Results Table:

Method	Average Precision	Runtime
Random	0.002	0.001
Harris w/o NMS	0.118	0.001
Harris w/ NMS	0.466	0.07
Alpha=0.04	0.457	0.06
Alpha=0.05	0.467	0.06
Val set of numbers of the bestmodel(From Gradescope)	0.466	0.06

Visualizations:





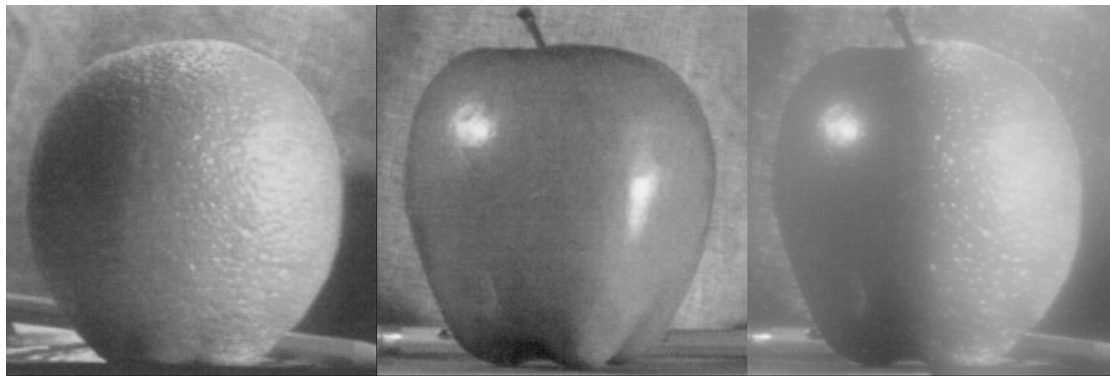
The implementation of the corner detection reached in general satisfying results. In relatively simple images, it is able to detect all of the corners. However, in complex scenes, my algorithm mis-labeled some of the corners. For example, as we can see in the front of the ECE Building, my implementation mislabeled the cross of the sidebar of the windows as corner, which is understandable. There indeed exists rapid intensity change at those points. I also missed some of the corners, for example, the top of the Beck man institute was mislabeled. The eventual results lies between 0.464 to 0.467. Also, even though I applied NMS, in the first image we can still notice multiple corners were marked at the corners of the cube.

Blending - Solution

1.Method Description:

First, to generate the stacks or pyramids. I applied Gaussian filter with different sigma values to the original images, this formed each level of my image stack for both images. I first set up an initial sigma, the relationship between the sigma we use and the level number is $\text{new_sigma} = \text{init_sigma} * (2^{**}\text{level}-1)$. For each newly generated level of the image stacks, I subtracted the last level of the this level to get the Laplacian pyramids for both of the image1 and image2. Using the similar methods with levels subtracting each other, I built a Gaussian Pyramids for the mask. Eventually I used the formula from page 230 to get the final pyramids, and add up every level of the pyramids to generate the blended image.

2.Oraple:



3.Blends of my own choice

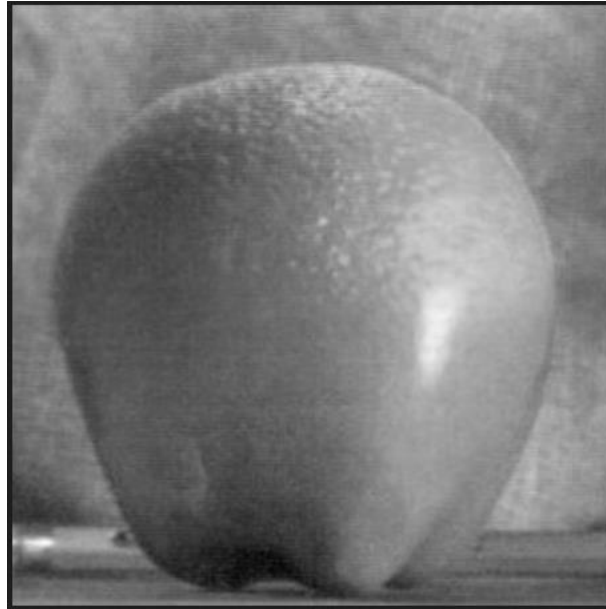
At first I tried the blending with the game concept art of my favorite video game, Red Dead Redemption, just to see if my blending algorithm works for other images that can be blended via the vertical seam, even though the two pictures are not as well-blended as the orange and apple images(They are quite different in shapes), I got relatively good results.



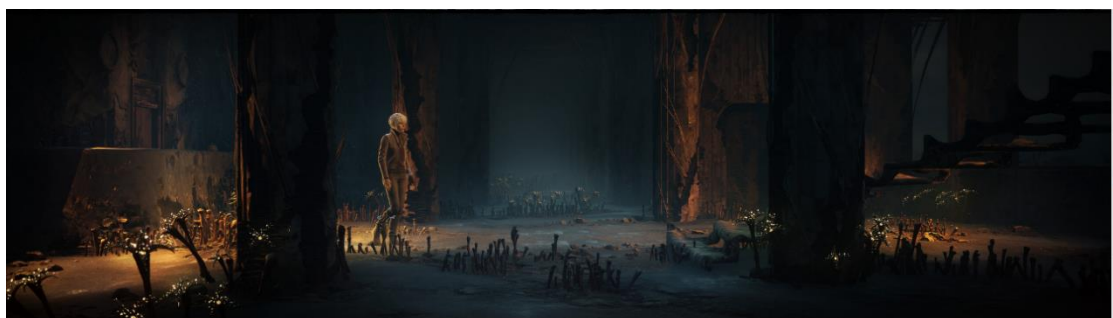
I did not make any modifications for this, except that I realized two images might

have different sizes, so I modified my code so that one image always have the same size with another.

To test the horizontal blending of two images, I used the image processing software to rotate the original image picture provided. This leads to the following blended images for orange and apples.



For further testing, I used two screenshots from video game MEDIUM, here are the results:





The two images are extremely similar by themselves (this game is pretty good! I strongly recommend), which results in a perfectly blended image.