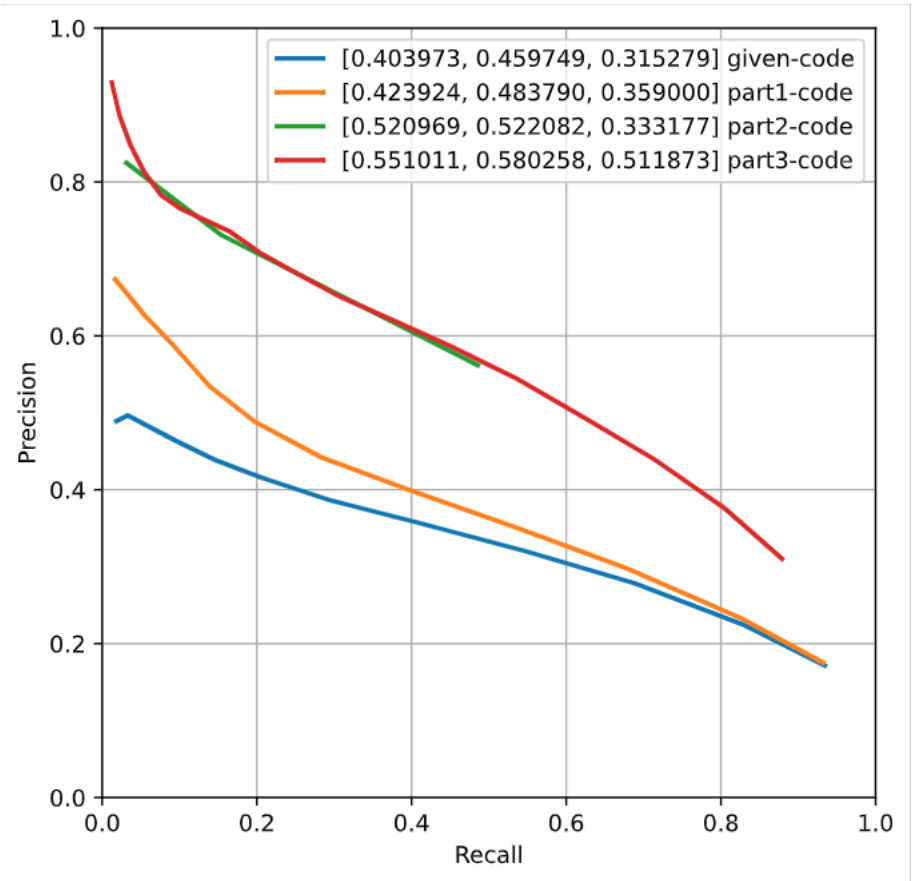
Contour Detection - Solution

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*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:

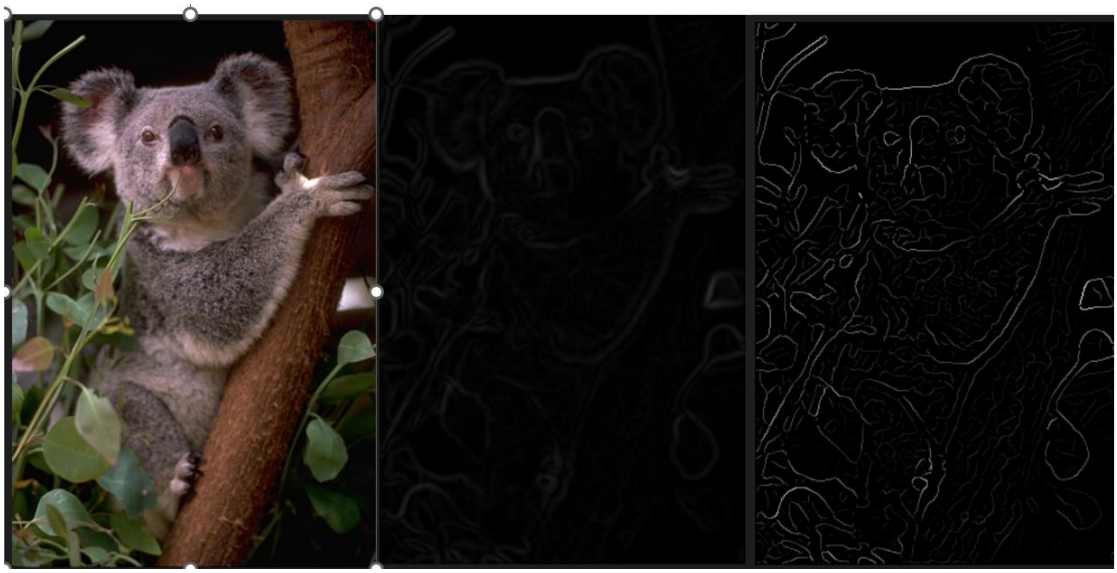


Results Table:

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

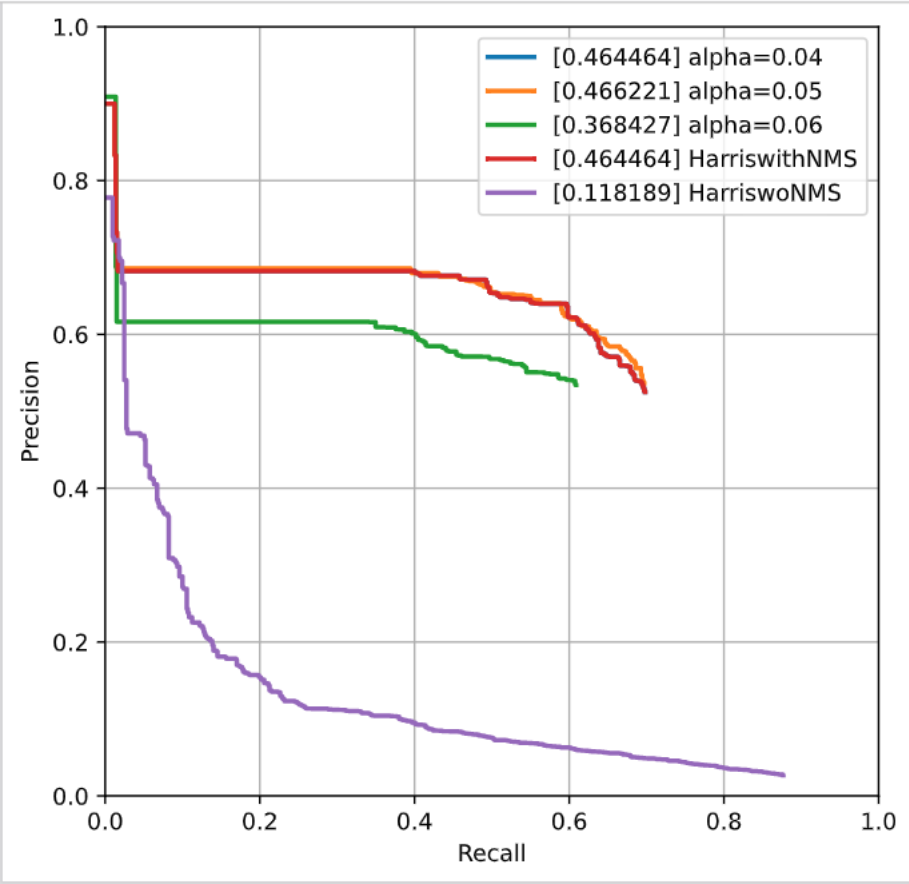
Method Description：

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*(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.

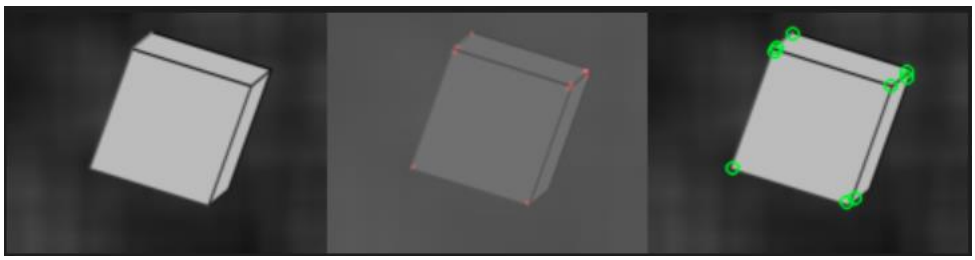
Precision Recall Plot:



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:



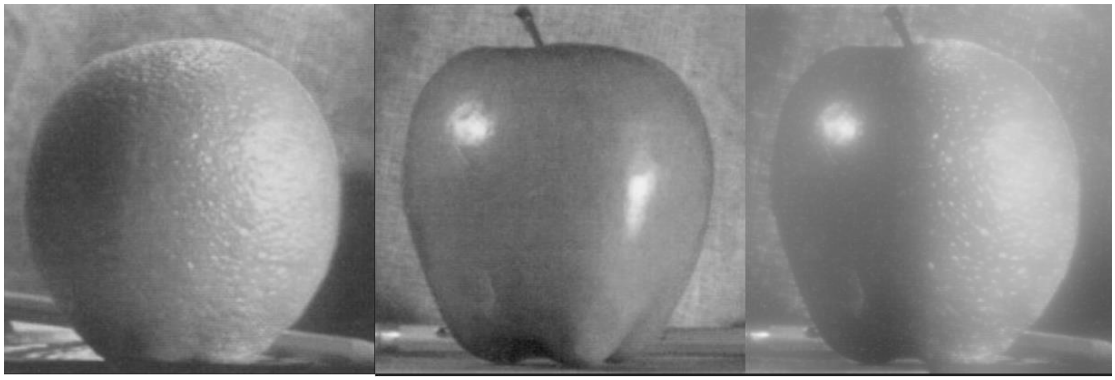
The implementation of the corner detection reached in general satisfying results. In relatively simple images, it is able to detect all of the corners. However, in complex scenes, my algorithm mis-labeled some of the corners. For example, as we can see in the front of the ECE Building, my implementation 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：

I used the Image Stack to generate the multiresolution blending at first time, but due to my poor implementation, I wasn't able to get satisfying results. So I changed to the Image Pyramids covered in class. I first generated Gaussian pyramids for both **img1** and **img2** using the **cv2.pyrDown** function from the OpenCV library. Next I continue to generate the Laplacian Pyramids for both images using the difference between the Gaussian pyramid at one level and the Gaussian pyramid at the next level. Then the Laplacian pyramids for both images are then blended together by concatenating the left half and right half of each level. Eventually, I used **cv2.py2up** function to upsample each level of the pyramids and blend them into one picture.

2.Oraple:



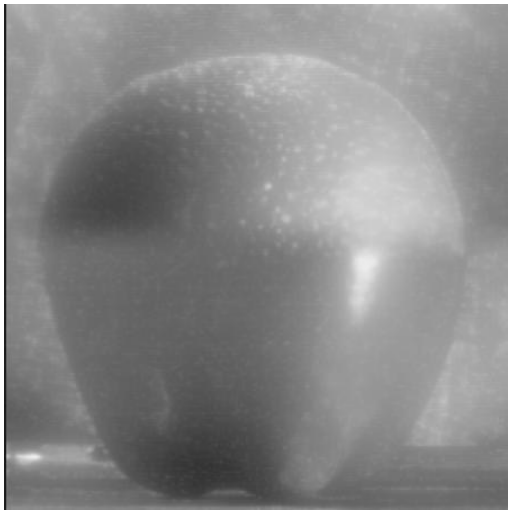
3.Blends of my own choice

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 thought the two pictures are not as well-blended as the orange and apple images, I got relatively good results.

I did not made 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 make the pictures can be blended through the horizontal lines, I modified the code when Laplacian pyramids of two images, instead of using the left and right half of the images, I used upper and lower half of the images. The following is what I got for Oraple:



To test the horizontal blending, I used two screenshots from video game MEDIUM, here are the results:

The two images are extremely similar by themselves, which leads to a perfectly blended image.