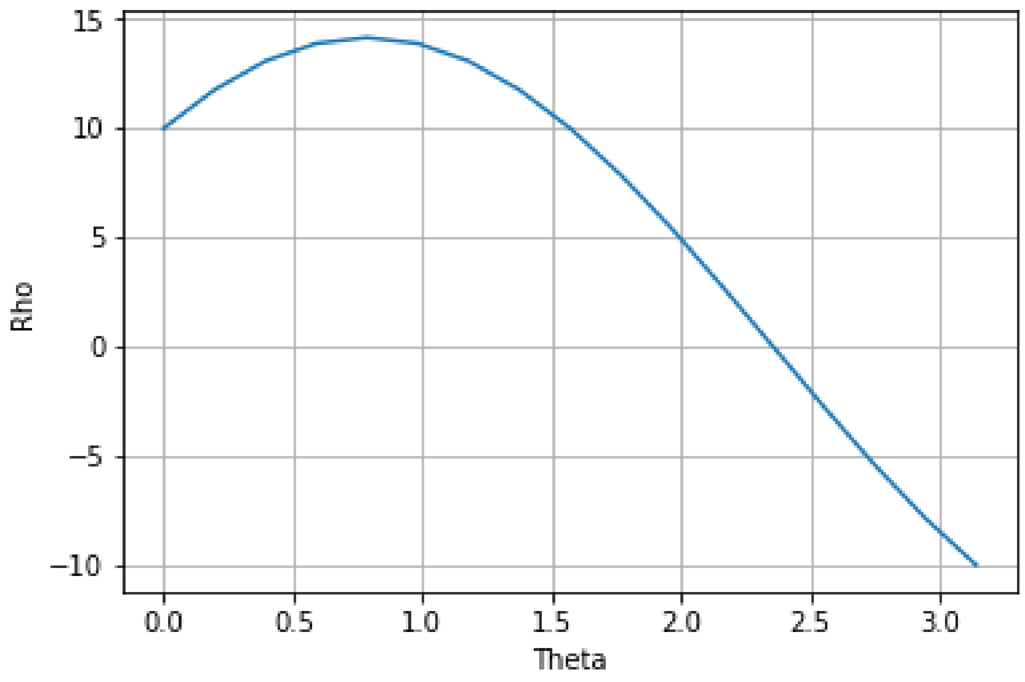
Theory Questions

1.

$$\rho = x * \cos(\theta) + y * \sin(\theta)$$
$$x * \cos(\theta) + y * \sin(\theta) = A * \sin(\varphi + \theta)$$
$$x * \cos(\theta) + y * \sin(\theta) = A * \sin(\varphi) * \cos(\theta) + A * \cos(\varphi) * \sin(\theta)$$
$$Amplitude \quad A = \sqrt{x^2 + y^2}$$
$$Phase \quad \varphi = \tan^{-1}\left(\frac{y}{x}\right)$$
$$\rho = \sqrt{x^2 + y^2} * \sin\left(\theta + \tan^{-1}\left(\frac{y}{x}\right)\right)$$

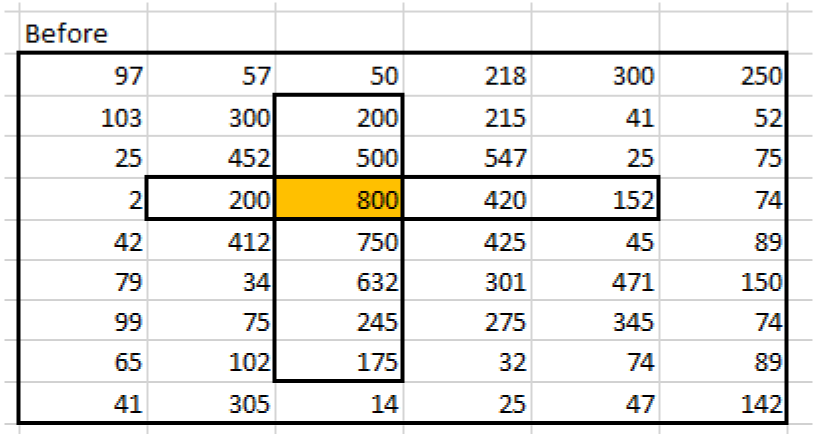
2. The range of theta is:
$$0 \le \theta \le \pi$$
When the upper left corner of the image is defined in the center of the image, the maximum value of rho with width W and height H is given by the following:
$$\rho_{MAX} = \sqrt{(W)^2 + (H)^2}$$
3. Sinusoid wave in Hough space for the point (10, 10) in the image with origin in upper left of the image.



Experiments

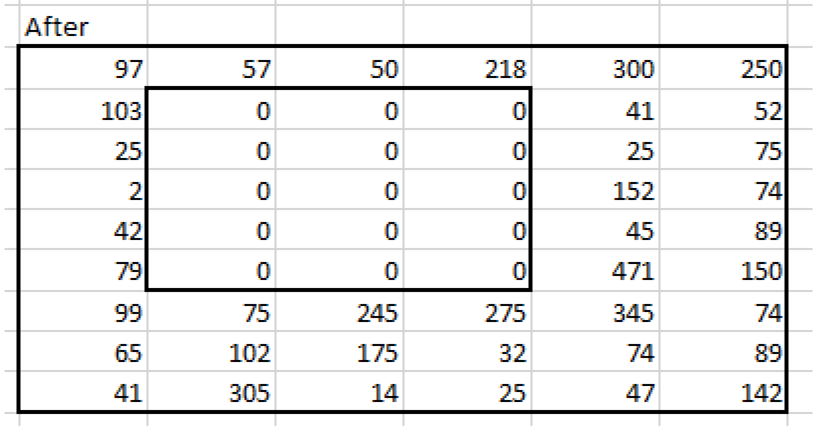
By varying parameters such as theta resolution, line count and sigma I was able to see better results for different images. By decreasing the theta resolution from pi/90 to pi/180 I was able to see better results in every image. For example, by making this change, both sides of the road in the road picture were now detected instead of only the one side it saw before. Although, with this change I actually lost detection of the horizon in that same photo. I noticed that with a smaller sigma, around 1, more vertical edges were detected in images such as the building and the home. Unfortunately, at a smaller sigma, I lost detection of both sides of the road. Images such as the side of the building and the road continued to improve with increasing the threshold, even past 0.3. Although, at this point, images like the hallway and cardboard boxes started to lose some defining lines.

The part of the algorithm that caused the most problems was implementing the non-maximum suppression. I decided to implement my own algorithm where it looks at the pixels around the maximum value and counts how far it takes to get to a pixel with intensity some percent of the maximum. I created a count for each direction (up, down, left, right) and taking the minimum value left/right and minimum value up/down I centered a mask with dimensions row offset x column offset and set all these values to 0. This can be seen below for a percent value of 10%:



Before

97	57	50	218	300	250
103	300	200	215	41	52
25	452	500	547	25	75
2	200	800	420	152	74
42	412	750	425	45	89
79	34	632	301	471	150
99	75	245	275	345	74
65	102	175	32	74	89
41	305	14	25	47	142



After

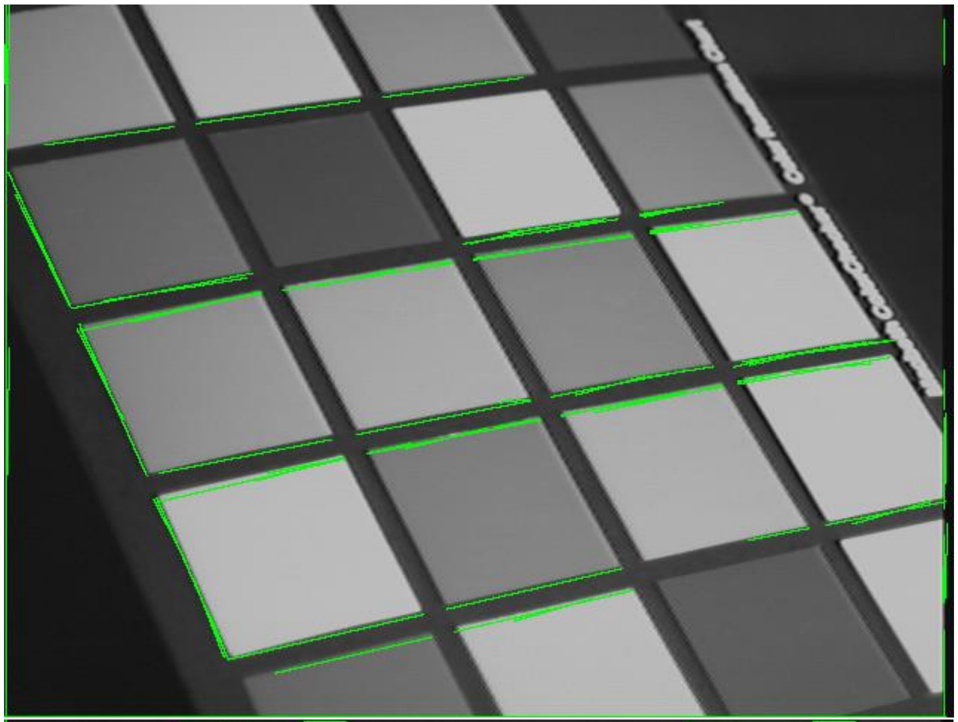
97	57	50	218	300	250
103	0	0	0	41	52
25	0	0	0	25	75
2	0	0	0	152	74
42	0	0	0	45	89
79	0	0	0	471	150
99	75	245	275	345	74
65	102	175	32	74	89
41	305	14	25	47	142

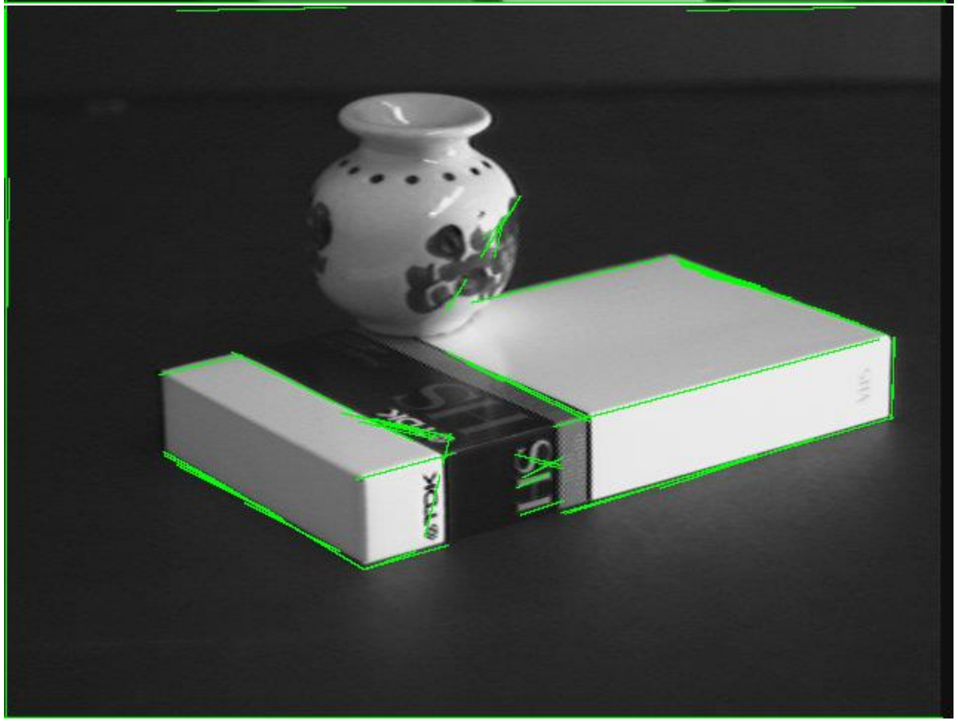
I had initially had the percent set to 10%, but I still noticed many duplicates in the lines on the images outputted. So I continued to decrease the percent, which effectively made more of the hit cell's surrounding neighbors 0. I had the idea to not take the minimum of right, left and up, down , but it wasn't nearly as robust and for some of the photos it would make the whole Hough transform output 0, meaning I deleted to many neighbors to the point when there was none left in the photo. Proof of this part of the algorithms potential to cause the most problems can be seen in the picture of the large office building. Quite frankly most of the results are bad for this image, and I believe it is because the lines are all clustered around the same area in the Hough Transform and my algorithm is deleting away some of the key lines since the magnitude of the pixels may never reach 10% of the maximum.

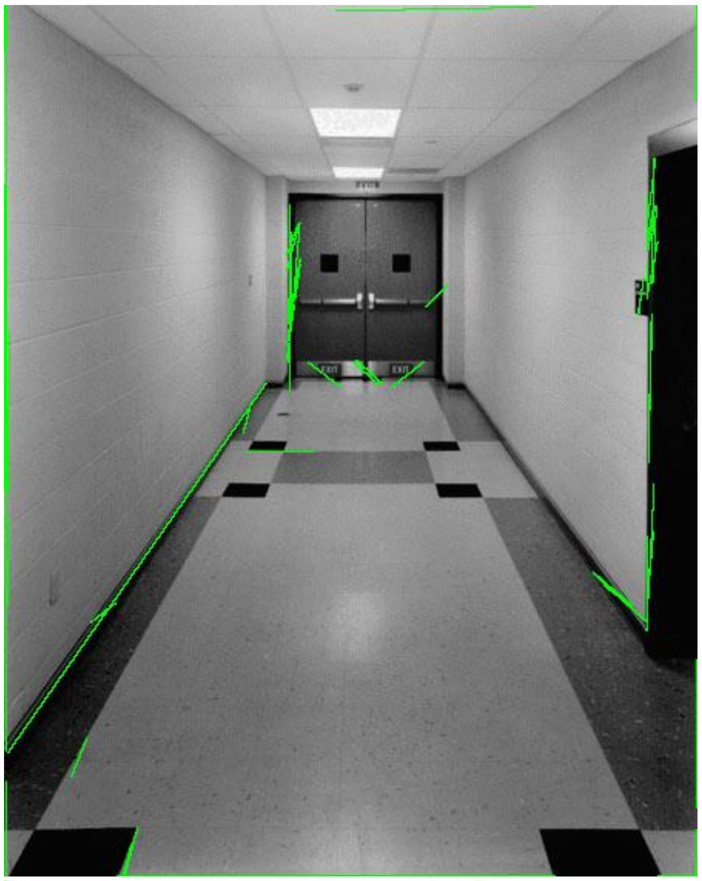
To improve performance, using the Fourier Transform could have saved a lot of computational power and time but unfortunately due to other time constraints I was not able to implement this. Also, to increase performance I had centered the origin for the Hough Transform to be in the center of the image, thus the Hough accumulator array H could be half the size since the maximum rho value would be cut in half. Unfortunately this didn't end up working since HoughLines.py is written assuming the origin is in the upper left corner of the image, as it is traditionally. One more thing that could have been done in myHoughTransform was to vectorize the operation using functions like np.outer. Due to my lack of experience in Numpy I couldn't get it working correctly, so it wasn't implemented

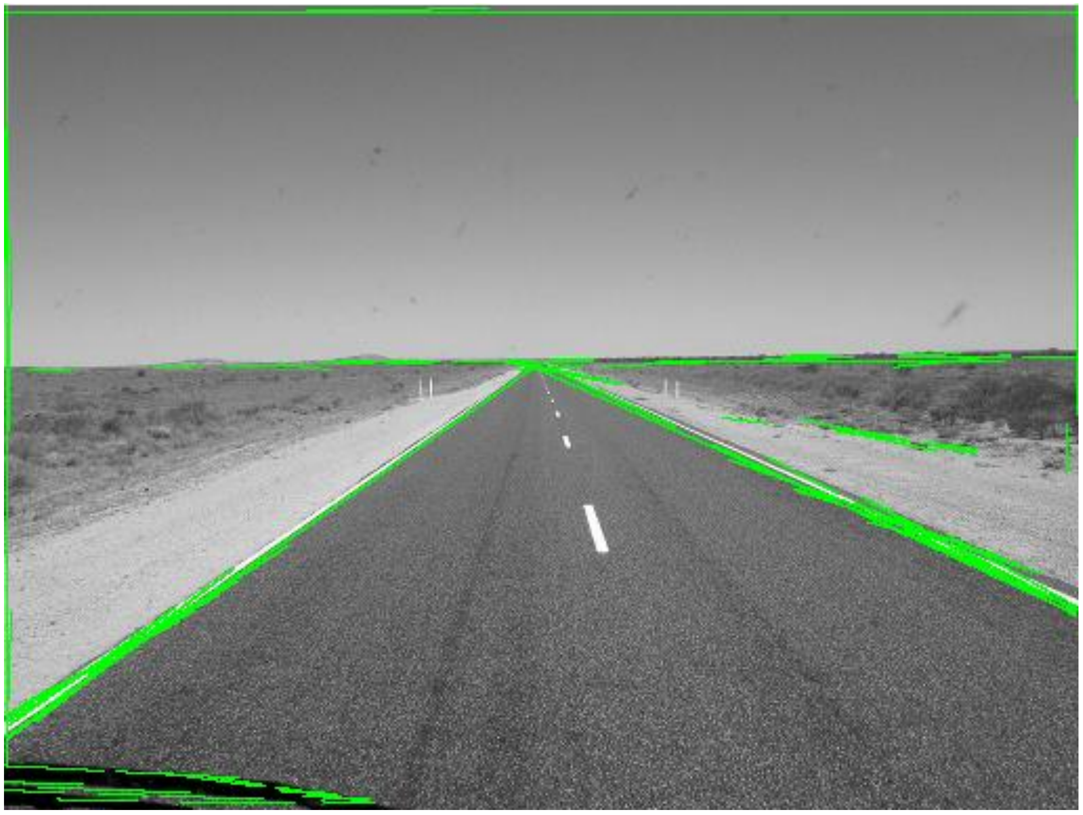
Image Results

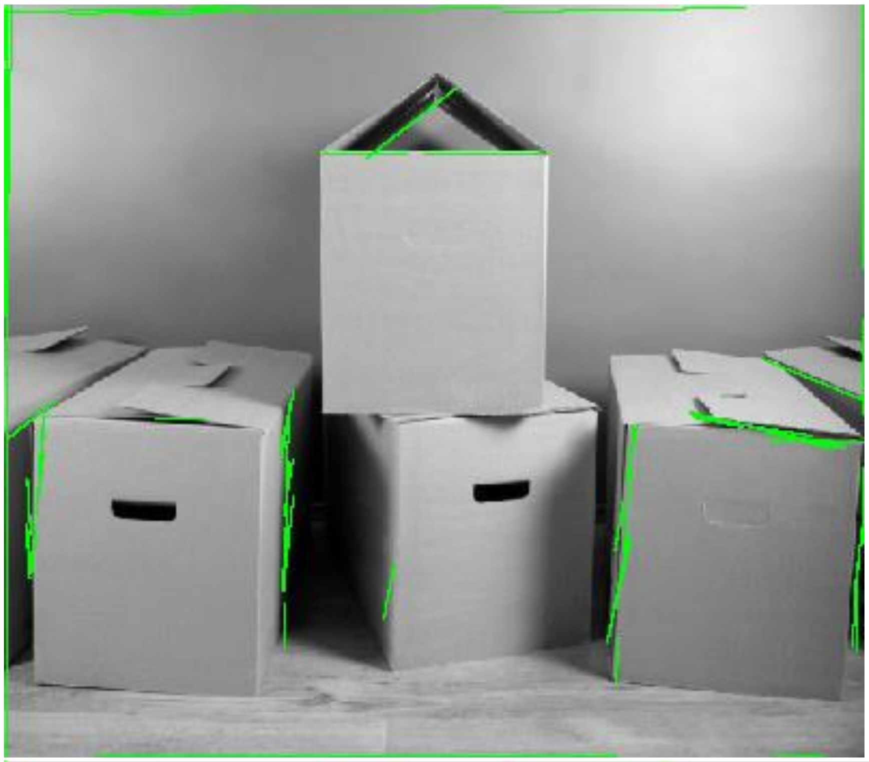
Final parameters: Sigma = 1.5, threshold = 0.3, rhoRes = 2.0, thetaRes = pi/180, nLines = 50.

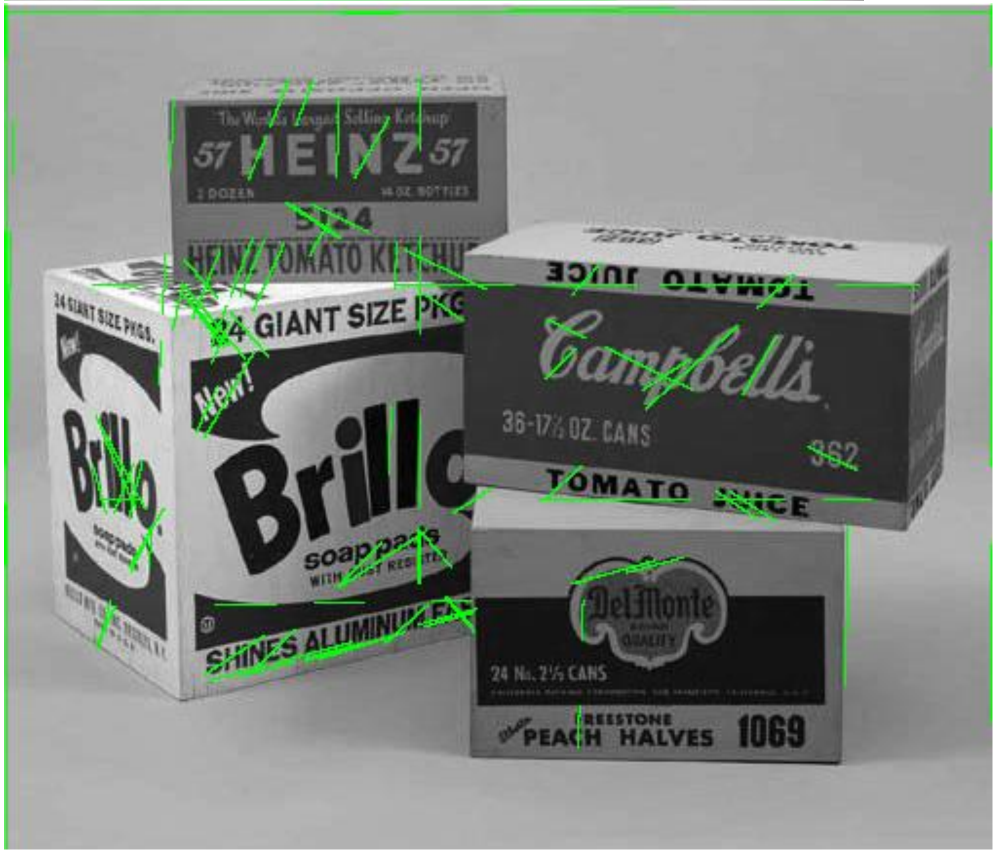


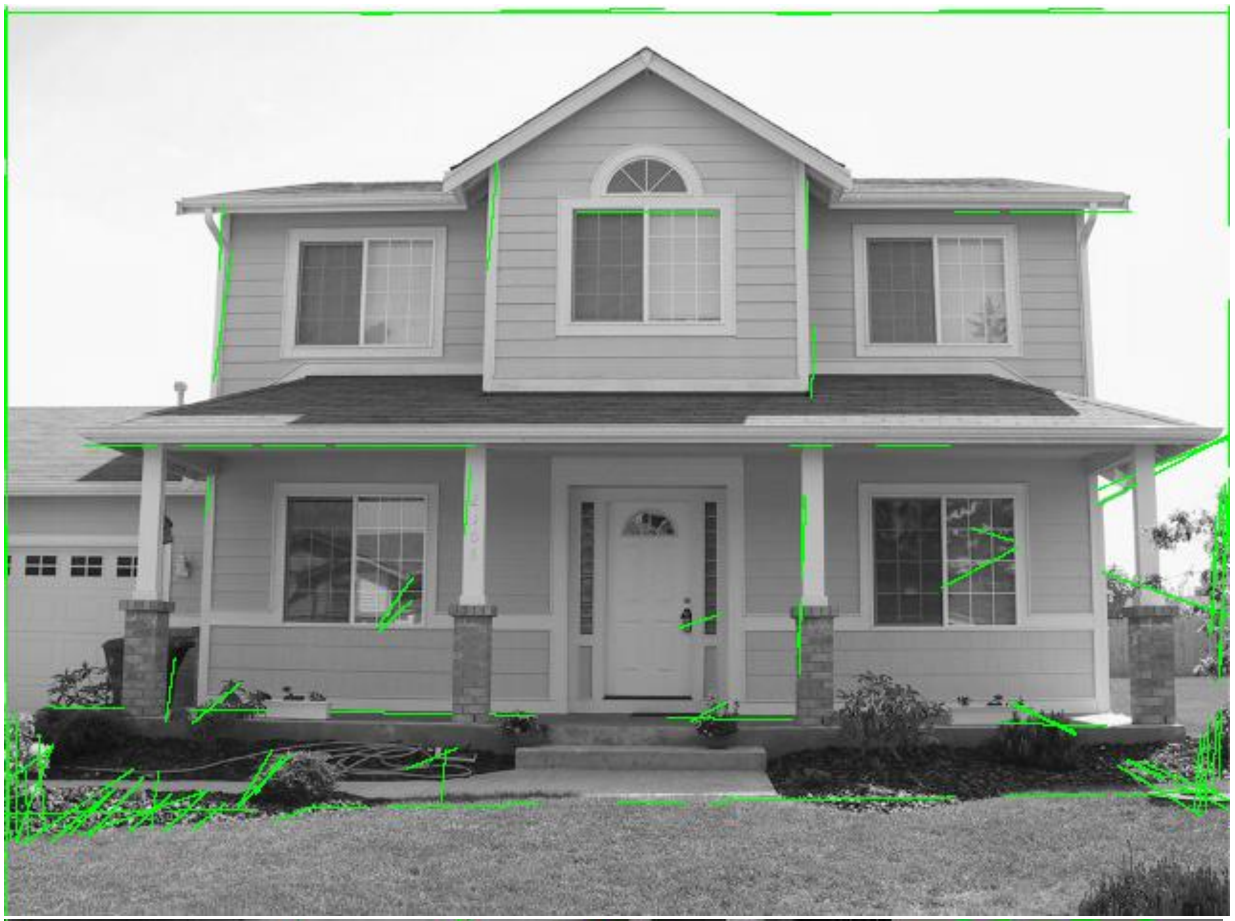


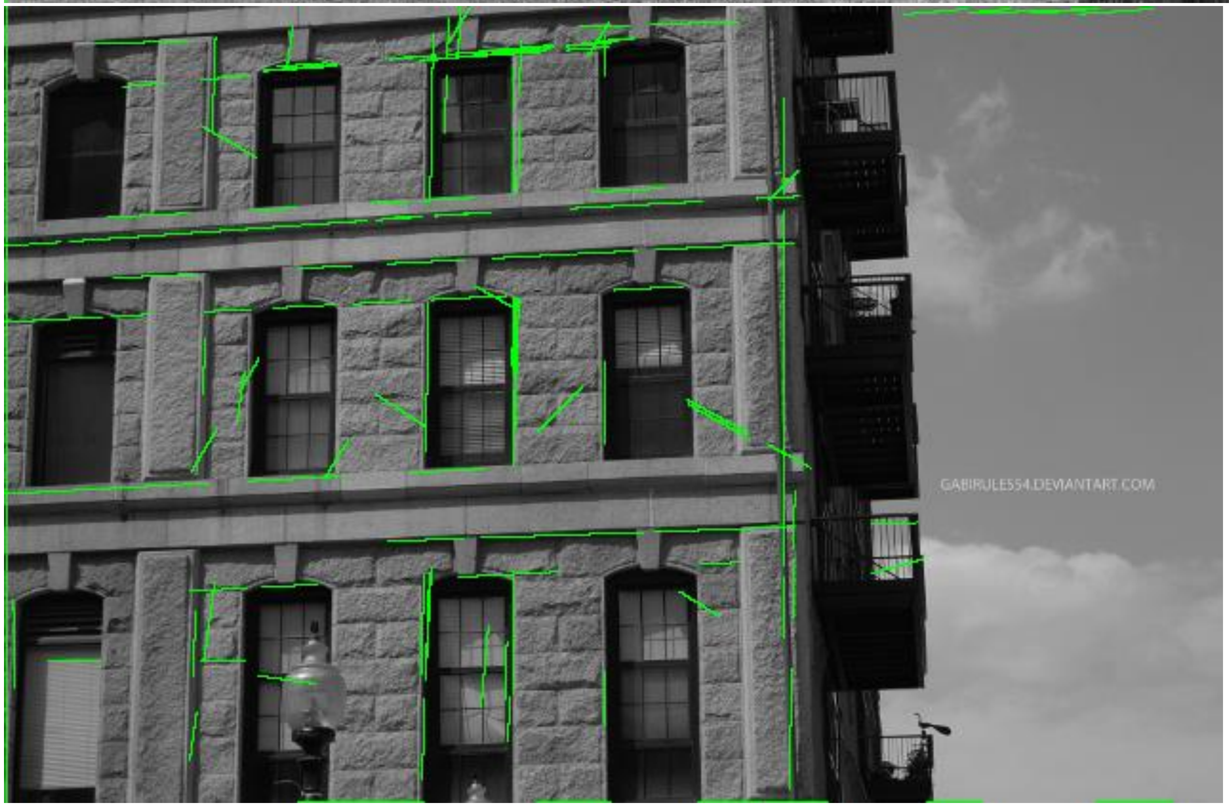


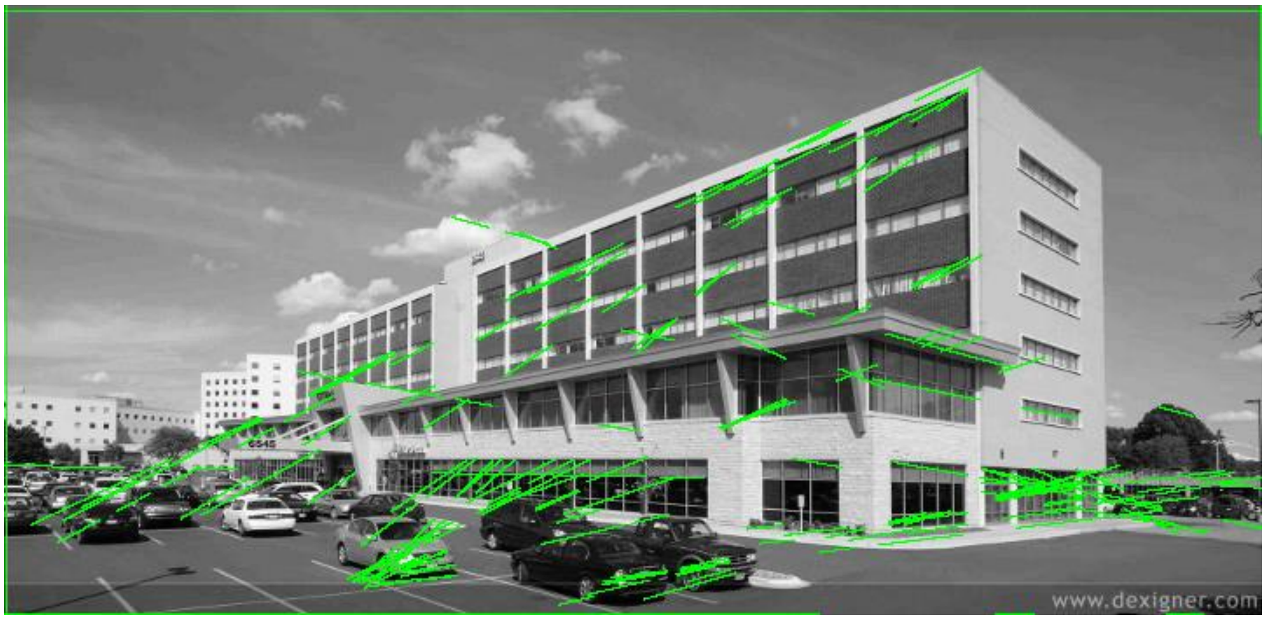




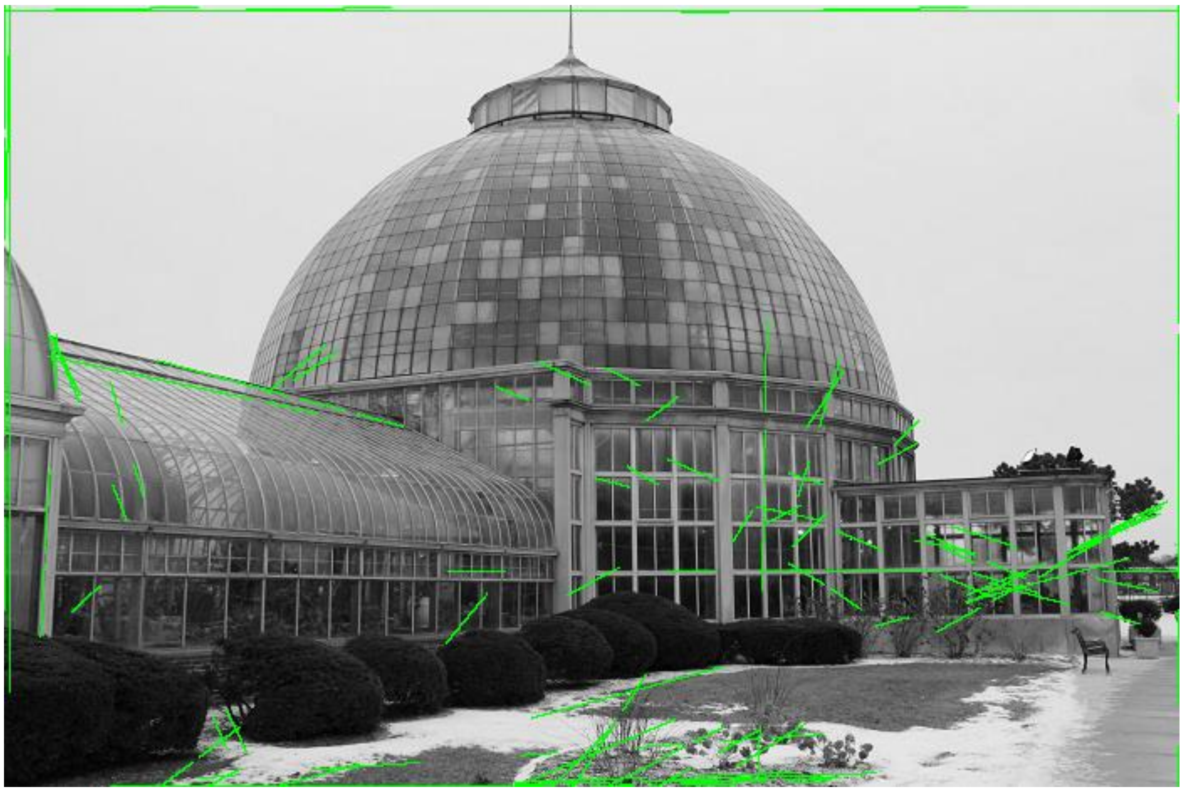

57 HEINZ 57
The World's Largest Selling Ketchup
2 DOZEN 14 OZ. BOTTLES
5124
HEINZ TOMATO KETCHUP
24 GIANT SIZE PKG
New!
Brillo
soap pads
SHINES ALUMINUM FAST
TOMATO JUICE
Campbell's
36-17½ OZ. CANS
362
Del Monte
24 No. 2½ CANS
FREESTONE
PEACH HALVES
1069

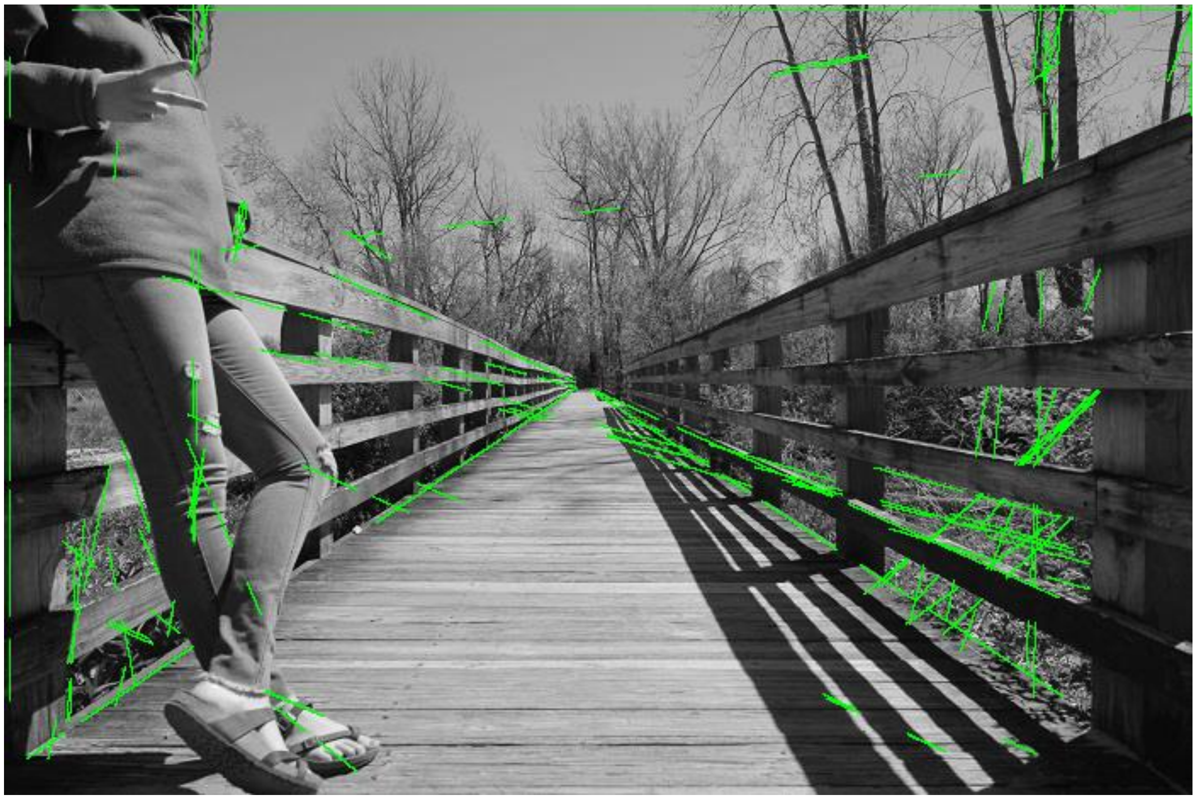


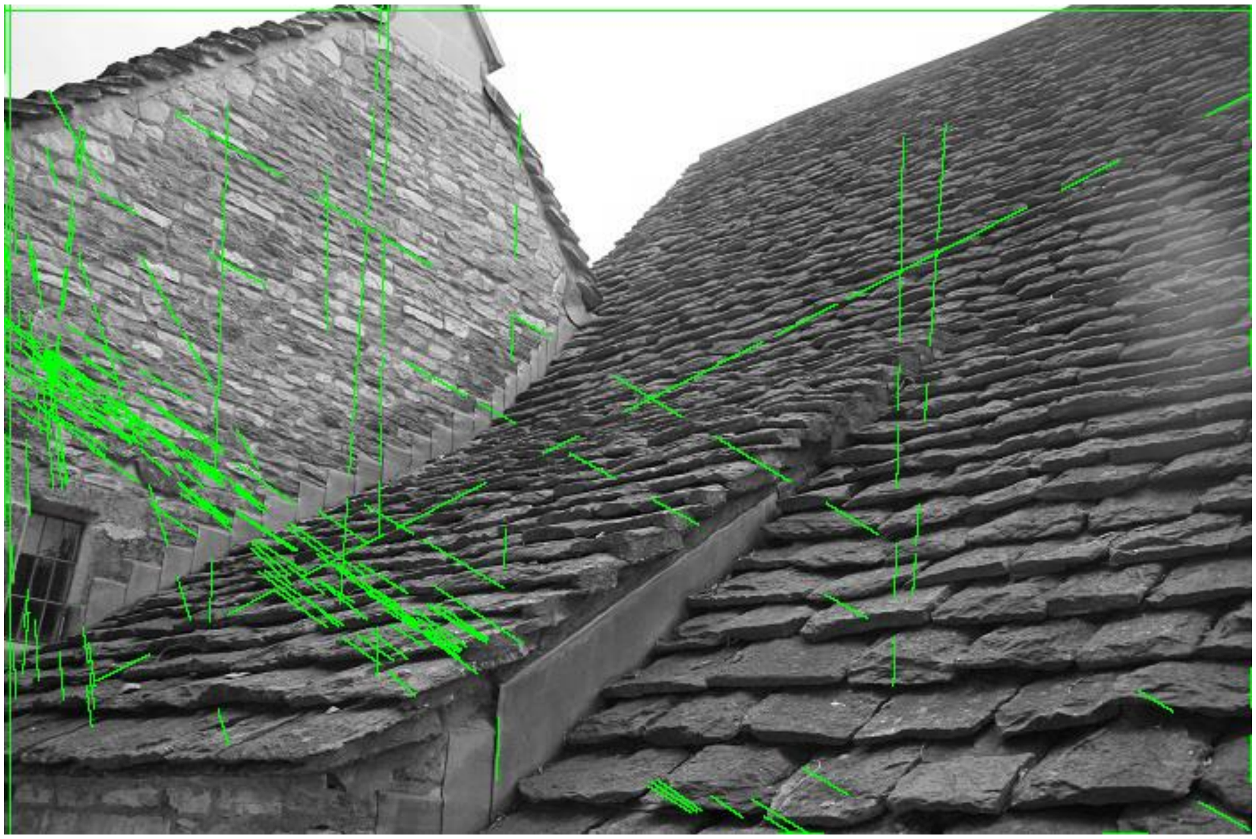

GABIRULE554.DEVIANTART.COM

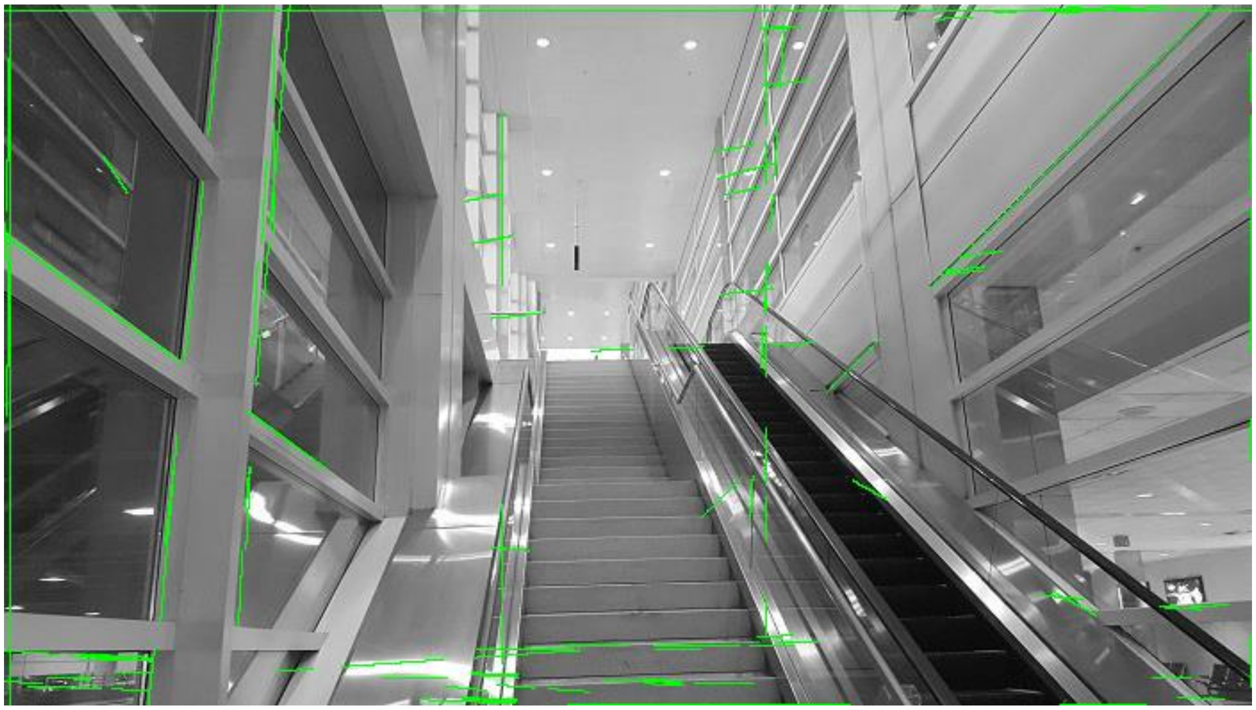

6545
www.dexigner.com

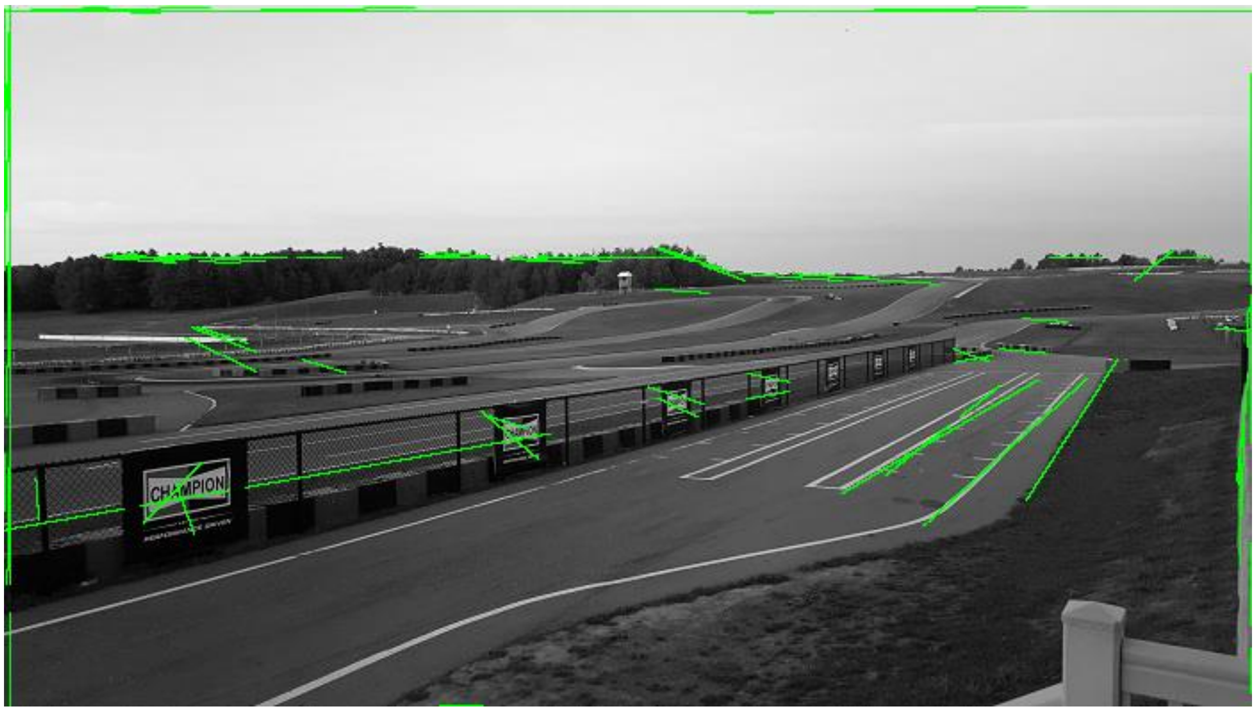
Extra Credit Photos










CHAMPION