BY: Ronak Kaoshik 17110068
Canny Edge Detector

This process from the input RGB image to the output binary image denoting the edges in the original image is explained below:

1. RGB to Grayscale:

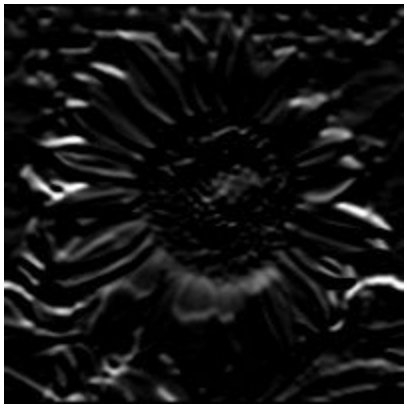
   The input image is converted to grayscale and the resulting image is as follows

2. A Gaussian filter is applied to this grayscale image to obtain a smooth image for further processing.

3. Sobel edge detection: A Sobel filter is applied in two different directions in order to record the abrupt intensity change in the respective directions.
   For x direction the change is as follows:

   For y direction the change is as follows:

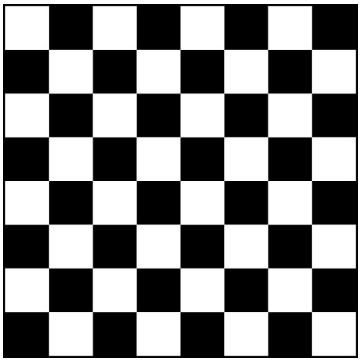


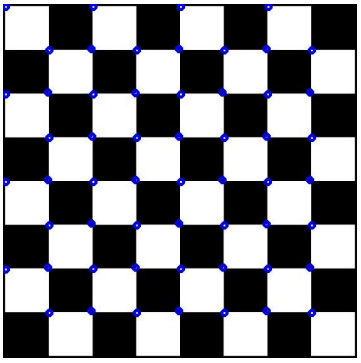
4. Non-maximal suppression: The magnitude of the above derivatives is calculated along with the direction of their pointing. A maximum point from these in the calculated direction is selected for later processing.
5. Hysteresis thresholding: A high and a low threshold is defined wherein a set of strong and weak edges are obtained by tuning the thresholds appropriately.
6. The weak edges are linked with the strong ones to get the final set of edges which are as shown below:

The output of the input canny detector with similar thresholds is as below:

The reason for OpenCV's Canny performing better than mine is due to the use of interpolation in OpenCV.

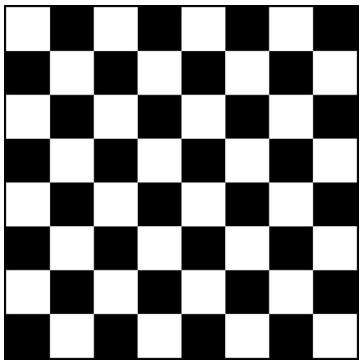
Harris Corner detection





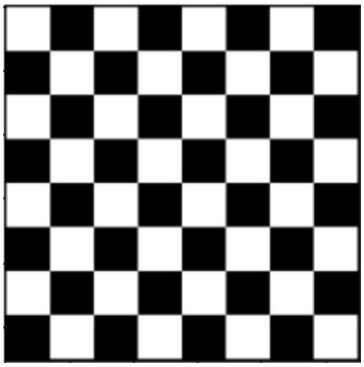
For the Corner detection using Harris' algorithm the following steps are followed

1. RGB to Grayscale

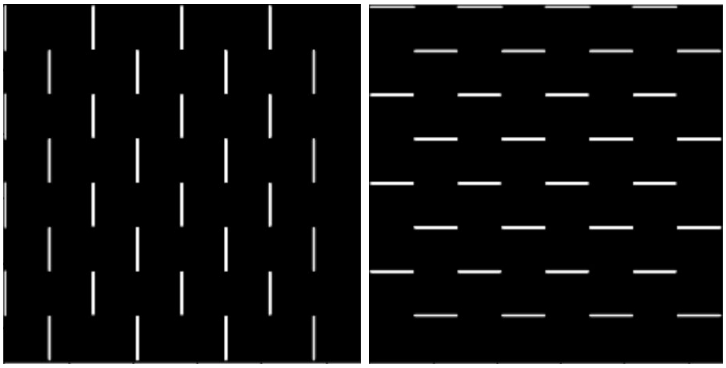


(Here it’s not evident due to the type of image)

2. Gaussian smoothing is applied on the grayscale image for further processing.

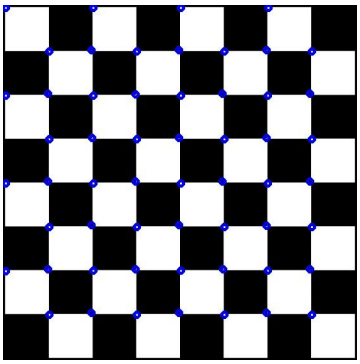


3. Derivatives are calculated in the x and y directions using sobel filtering and following are the respective outputs.



4. Double derivatives are calculated for the matrix as per Harris' algorithm and Gaussian blur is applied to avoid abrupt noises that can be miscounted as corners. The 'R' metric is calculated for all the pixels as per Harris' algorithm which later decides the classification for corners.
5. Once the R value for each pixel is obtained, by repetitive tuning the appropriate value for corners is found. (In my case the value was around 40000). These pixels are then

plotted using non-maximal suppression for chunks of pixels to avoid concentrations of pixels around corners. The output is as follows:



Reference:

1. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html
2. https://en.wikipedia.org/wiki/Canny_edge_detector
3. https://docs.opencv.org/3.4/d4/d7d/tutorial_harris_detector.html
4. https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html