COURSE: CS5404 - INTRODUCTION TO COMPUTER VISION

INSTRUCTOR: DR. ZHAOZHENG YIN

ASSIGNMENT #2:

HARRIS CORNER DETECTION

Connor Coward

Missouri University of Science and Technology

October 15, 2016

Contents

OBJECTIVE

The objective of this assignment is introduce some basic concepts of computer vision programming by writing an implementation of the Harris corner detection algorithm.

IMPLEMENTATION

The algorithm was implemented in Python with help from the Numpy and OpenCV libraries for performing basic image manipulation tasks such as importing and exporting images, performing convolution, drawing, etc. Note: there are many libraries available which would perform the required tasks in just one or two lines of code. In the spirit of this assignment, those libraries were not used. The full code is listed in the Code Listings section at the end of this document.

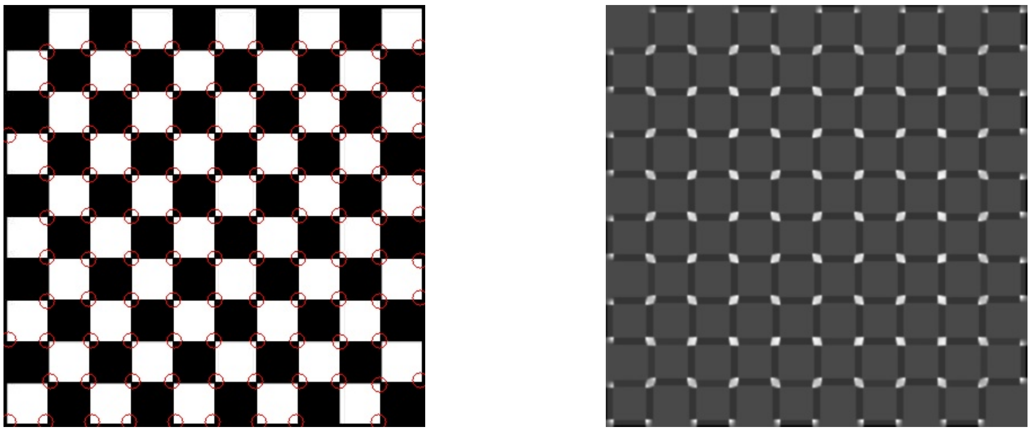
For all of the images, the following initial settings were used:

- **Gaussian Blur Sigma: 1.2**
- **Harris Neighborhood Size: 7**
- **Non-Maximum Suppression Radius: 15**

Note: The ideal number of corners was manually adjusted to the approximate amount of corners featured in each image.

RESULTS

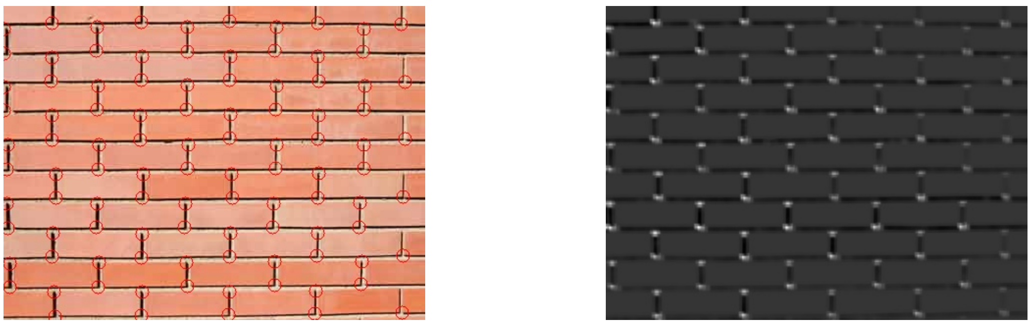
The program is overall very effective at detecting corners, although the effectiveness depends highly on the choice of input parameters. For ideal images, such as the checkerboard image, the results are very good. The checkerboard image with detected corners indicated and Harris R-score map are shown in Figure 1. In the R-Score image, the regions of high R-score are clearly visible as white blobs. The regions between the squares of the checkerboard patterns are neutral gray, indicating a value of zero, or nearly zero. Finally, the areas where there is a vertical or horizontal edge in the checkerboard pattern, the R-score value is negative.



(a) Checkerboard image with detected corners **(b)** R-Score image for the checkerboard pattern

Figure 1: Checkerboard image corner detection results

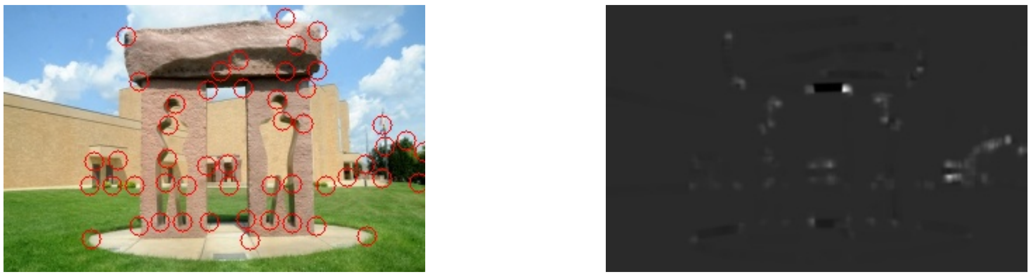
The next-best results came from the brick wall image. This image also features sharp, consistent corners in an even pattern, making it ideal for corner detection. However, there are shadows, noise, etc. which causes a few errors in the algorithm. For instance, some of the corners are not accurately centered directly on the corners in the image. Overall, though, the algorithm is able to correctly identify the vast majority of the corners featured in the image. The results for the brick wall image are shown in Figure 2.



(a) Brick wall image with detected corners **(b)** R-Score image for the brick wall image

Figure 2: Brick wall image corner detection results

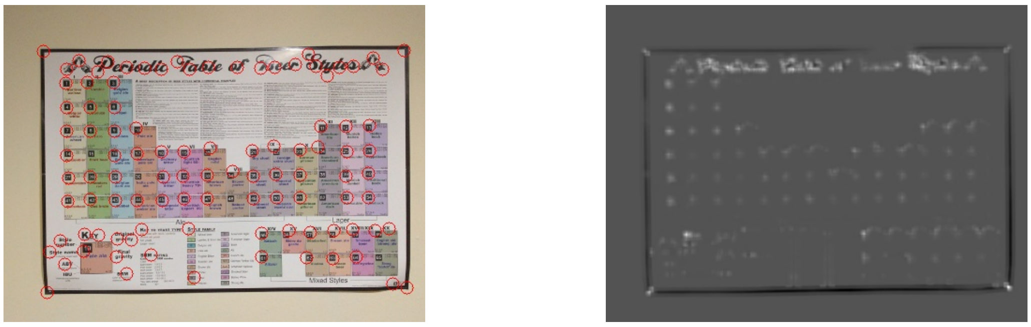
Images such at the granite statue image with many complex features such as grass, clouds, curves, etc. has worse results than the checkerboard or brick wall image. The results of the granite statue image are shown in Figure 3



(a) Granite statue image with detected corners **(b)** R-Score image for the statue

Figure 3: Statue image corner detection results

Finally, a fourth image of a "Periodic Table of Beer" poster was included. This image has small detailed features, but also many prominent corners, such as the corners on the edge of the poster. The results for the periodic table poster are shown in Figure 4.



(a) Periodic table image with detected corners **(b)** R-Score image for the periodic table

Figure 4: Statue image corner detection results

Design Decisions

Gaussian Kernel Size

To apply a Gaussian blur to the image, the size of the Gaussian kernel had to be determined. For this determination, the size of the kernel is calculated by multiplying the given sigma value by a factor of three. With a factor of three, this means that the sum of the values in the kernel total to 0.997, or 99.7% of the ideal, theoretical value. This is certainly good enough for almost all applications. Note: since the kernel must have odd dimensions, the calculated size was rounded to the nearest odd number.

Second Gaussian Blur

During development, it was observed that the R-score image often had splotchy, uneven patches where corners were located. Because corner locations are determined based on the absolute maximum value present in the R-score map, this unevenness resulted in less-accurate detected corner positions. A second Gaussian blur operation was included to smooth the high-frequency noise from the R-Score map prior to performing non-maximum suppression. This seems to result in corner locations which are more accurate to the true location of the corners in the image.

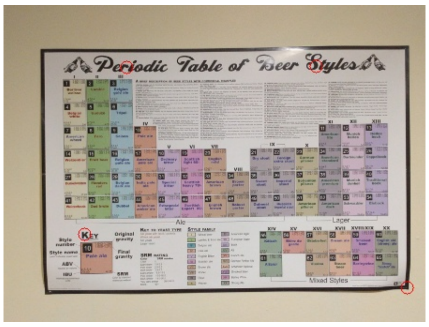
OBSERVATIONS

Contrast/Color Problems

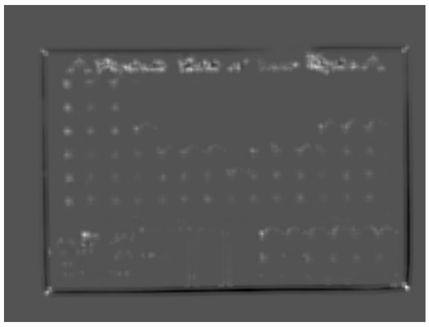
In the granite statue image, the corner detection algorithm missed a few key corners which a human can easily identify. For instance, the top right corner of the building in the background is a sharp 90-degree corner, yet the algorithm was not able to identify it as one of the "best" corners. In the R-Score image, no point is visible where the building corner occurs. This is most-likely due to the low contrast between the building and background sky. This effect is especially problematic because the image is converted to grayscale. The blue sky and white clouds in the background are clearly distinct from the pale-yellow/beige building color, however when converting to grayscale, that information is lost. For more accurate corner detection, perhaps the algorithm could be run on each channel (for example, reg, green, blue) individually, then combined into a final corner detection result.

Rectangle Corner Identification

One useful application of corner detection would be identifying the 4 corners of a square or rectangle. To test this application, I reduced the desired number of corners "M" down to 4 and ran the program on the periodic table of beer image. The results are shown in Figure 5.



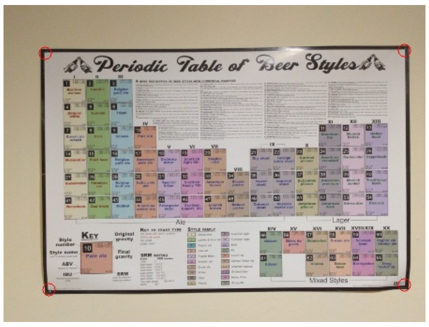
(a) Periodic table image with detected corners



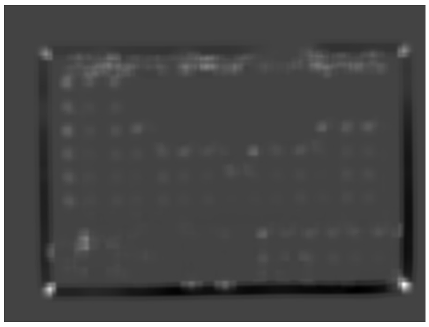
(b) R-Score image for the periodic table

Figure 5: Periodic table image with poor settings, corners not detected

Note that only one of the corners of the poster was accurately identified. To improve the detection rate, it was found that the Harris neighborhood size and Gaussian sigma needed to be increased significantly. By increasing the neighborhood size, smaller features such as the text on the poster are ignored, and the large corners on the edges of the poster are identified. Figure 6 shows the results with a Gaussian sigma of 2.0 and a neighborhood size of 15.



(a) Periodic table image with detected corners



(b) R-Score image for the periodic table

Figure 6: Periodic table image with improved settings to reject small details

Choosing Parameters

Gaussian Sigma

The Gaussian blur sigma value is important for rejecting high-frequency noise in the image in order to prevent the corner detection algorithm from giving false detections. Blurring leaves the large features in tact while removing the small, detailed features. For detecting a few large corners, a large amount of Gaussian blur is needed. For detecting many, or small corners, a small amount of blur is required.

Harris Neighborhood

The Harris neighborhood size parameter is related to the the amount of Gaussian blur. For instance, a huge amount of blur with a small Harris neighborhood is not effective because the intensity gradient across the neighborhood patch would be too small to give accurate results. the Harris neighborhood size should be large enough that it can encompass the entire corner and sounding edges comfortably, but not so large that it also overlaps with other nearby corners.

Non-Maximum Suppression (NMS) Radius

The Non-Maximum Suppression (NMS) radius is the distance around each detected corner to suppress the R-value to prevent multiple detections of the same corner. This value must be large compared to the Harris neighborhood size in order to fully suppress the blob around each corner. However, it must also be small enough that any nearby corners are still left intact.

CODE LISTINGS

Listing 1 shows the file detectHarrisCorners.py. This file is where the actual corner detection happens. It contains a function called detectHarrisCorners() which accepts the parameters needed to calculate the corner locations and returns the corner locations and the R-score image.

Listing 1: detectHarrisCorners.py

```
import cv2
import numpy as np

def detectHarrisCorners(image, gaussianSigma, neighborhoodSize, nmsRadius, desiredNumberOfCorners):
    # Define constants
    K = 0.05
    NMS_GAUSSIAN_BLUR_SIGMA = 2
    NMS_GAUSSIAN_BLUR_SIZE = 5

    # Pre-Calculate critical parameters
    gaussianBlurSize = int((gaussianSigma*3)/2)*2 + 1 # Make kernel size 3 times the standard deviation -> 99.7%

    # Convert to grayscale, 32-bit float
    img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    img = img.astype(np.float32) / 255

    # Apply the gaussian blur
    Kx = cv2.getGaussianKernel(gaussianBlurSize, gaussianSigma)
    Ky = np.transpose(Kx)
    img = cv2.sepFilter2D(img, -1, Kx, Ky)

    # Calculate gradients
    Kx = np.array([[-1, 0, 1]])
    Ky = np.array([[1], [0], [-1]])
    Gx = cv2.filter2D(img, -1, Kx)
    Gy = cv2.filter2D(img, -1, Ky)

    # Calculate products of derivatives (Gx^2, GxGy, and Gy^2)
    Gx2 = np.square(Gx)
    GxGy = np.multiply(Gx, Gy)
    Gy2 = np.square(Gy)

    # Compute sums of products of derivatives
    Sx2 = cv2.boxFilter(Gx2, -1, (neighborhoodSize, neighborhoodSize), normalize=False)
    Sxy = cv2.boxFilter(GxGy, -1, (neighborhoodSize, neighborhoodSize), normalize=False)
    Sy2 = cv2.boxFilter(Gy2, -1, (neighborhoodSize, neighborhoodSize), normalize=False)

    # Calculate the Harris R score
    detH = np.multiply(Sx2, Sy2) - np.square(Sxy)
    traceH = Sx2 + Sy2
    R = detH - K*np.square(traceH)
    # Do another Gaussian Blur to filter out noise before NMS
    Kx = cv2.getGaussianKernel(NMS_GAUSSIAN_BLUR_SIZE, NMS_GAUSSIAN_BLUR_SIGMA)
    Ky = np.transpose(Kx)
    Rblur = cv2.sepFilter2D(R, -1, Kx, Ky)

    # Perform non-maximum suppression (NMS)
    corners = []
    while len(corners) < desiredNumberOfCorners:
        _, _, _, maxLoc = cv2.minMaxLoc(Rblur)
        corners.append(maxLoc)
        cv2.circle(Rblur, maxLoc, nmsRadius, (0, 0, 0), thickness=-1) # Suppres all pixels in NMS RADIUS



    # Return the image and corner locations
    return corners, Rrs.py" 58L, 2023CTraceback (most recent call last):
```

Listing 2 shows the file main.py. This file is a driver/test file for detectHarrisCorners.py. Main.py loads detectHarrisCorners and uses the function to detect corners in the file given as a command line argument.

Listing 2: main.py

```
import cv2
import sys
import numpy as np
from detectHarrisCorners import detectHarrisCorners

NUM_ARGUMENTS = 8
USAGE_MESSAGE = sys.argv[0] + " <inputImagePath> <outputImagePath> <rscoreImagePath> \
    <gaussianSigma> <neighborhoodSize> <nmsRadius> <desiredNumberOfCorners>"

if len(sys.argv) != NUM_ARGUMENTS:
    print USAGE_MESSAGE
    sys.exit(-1)

# Parse the input files
inputImagePath         = sys.argv[1]
outputImagePath        = sys.argv[2]
rscoreImagePath        = sys.argv[3]
gaussianSigma          = float(sys.argv[4])
neighborhoodSize       = int(sys.argv[5])
nmsRadius              = int(sys.argv[6])
desiredNumberOfCorners = int(sys.argv[7])

# Read the input image from file
image = cv2.imread(inputImagePath)

# Run the harris corner detection algorithm
corners, R = detectHarrisCorners(image, gaussianSigma, neighborhoodSize, nmsRadius, desiredNumberOfCorners)
```

```
# Normalize the R-Score values for easier viewing
R = R - np.amin(R)
R = 255 * R / np.amax(R)
R = R.astype(np.uint8)

# Print the corner coordinates
print "Corner coordinates:"
for corner in corners:
    print corner

# Draw the corner locations on the original image
for c in corners:
    cv2.circle(image, c, 7, (0, 0, 255), thickness=1)

# Display the original image & R-Score image
cv2.imshow("Original image with corners", image)
cv2.imshow("Harris R-Score Image", R)
cv2.waitKey(0)

# Save the images to files
print "Saving images..."
cv2.imwrite(outputImagePath, image)
cv2.imwrite(rscoreImagePath, R)

# Exit
print "Done! Exiting..."
cv2.destroyAllWindows()
```