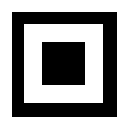
ECE 5554 SU22 – Dr. Jones – HW2

**Part 1:**

****

**Part 2b:**

Filename: QR\_A.png

(Row, Col, Intensity): (298, 34, 0.70864534)

(Row, Col, Intensity): (292, 262, 0.70212114)

(Row, Col, Intensity): (527, 40, 0.69987273)

Filename: QR\_B.png

(Row, Col, Intensity): (683, 492, 0.83787876)

(Row, Col, Intensity): (669, 885, 0.8826453)

(Row, Col, Intensity): (1075, 512, 0.83638155)

Filename: QR\_C.png

(Row, Col, Intensity): (94, 16, 0.41122884)

(Row, Col, Intensity): (60, 288, 0.49551252)

(Row, Col, Intensity): (422, 19, 0.40261316)

Filename: QR\_D.png

(Row, Col, Intensity): (174, 230, 0.8105585)

(Row, Col, Intensity): (229, 1277, 0.81091094)

(Row, Col, Intensity): (1221, 175, 0.8120636)

Filename: QR\_E.png

(Row, Col, Intensity): (348, 366, 0.50402844)

(Row, Col, Intensity): (346, 581, 0.49120975)

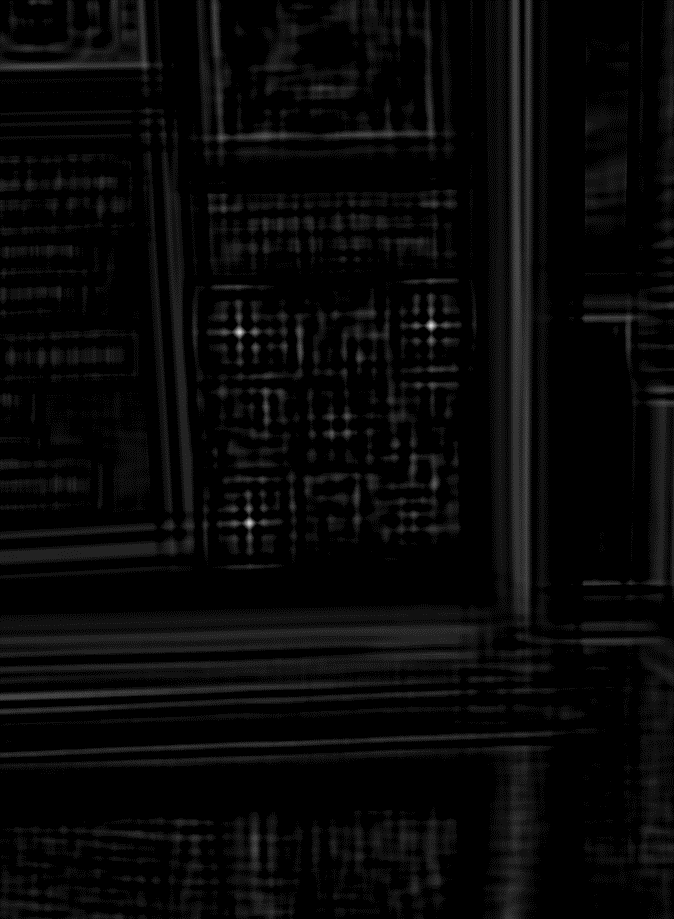
(Row, Col, Intensity): (580, 370, 0.48469767)

**Part 2c:**

matchTemp\_reults\_QR\_A.png



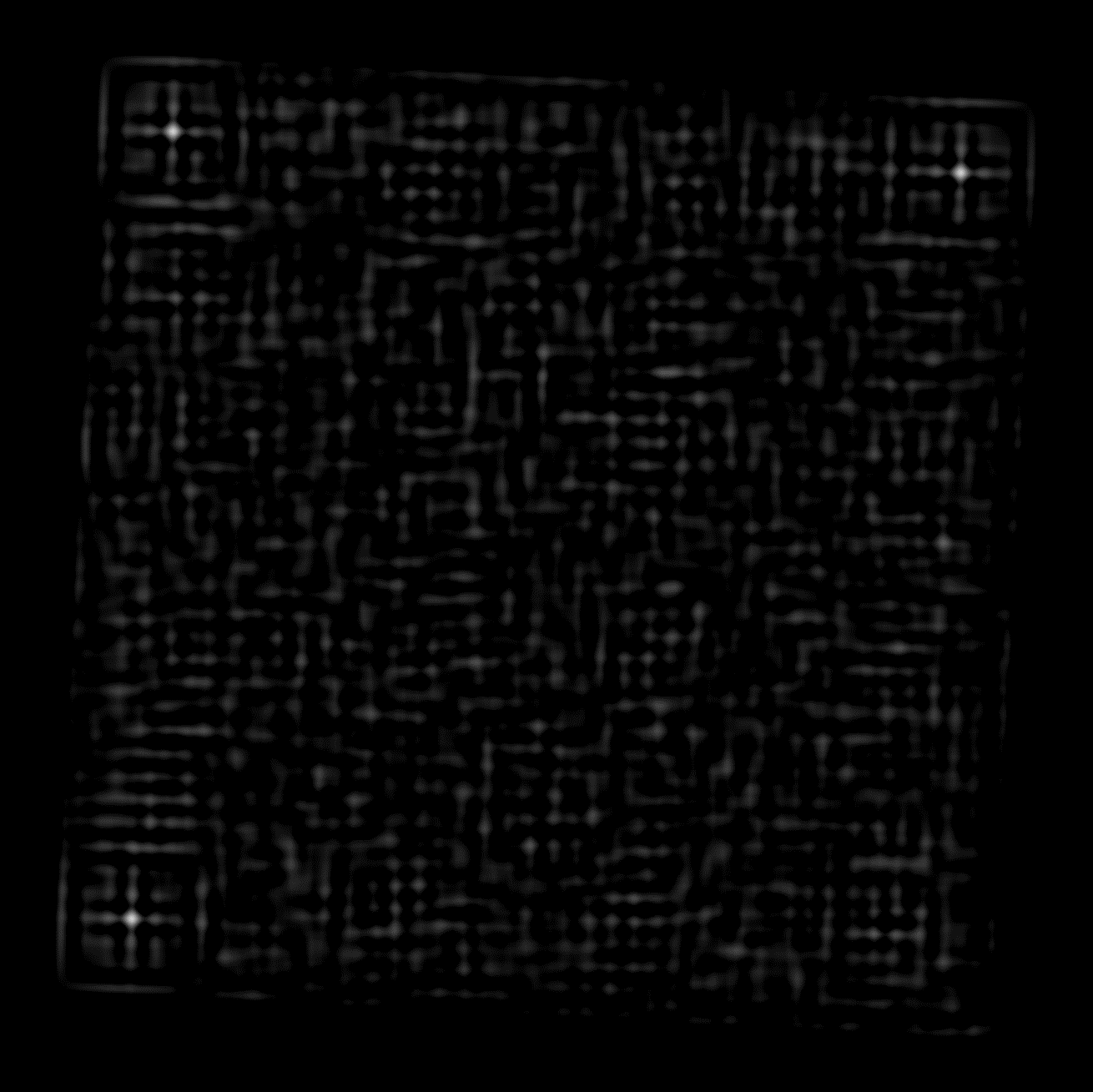
matchTemp\_reults\_QR\_B.png



matchTemp\_reults\_QR\_C.png



matchTemp\_reults\_QR\_D.png



matchTemp\_reults\_QR\_E.pngA picture containing text, black, white

Description automatically generated

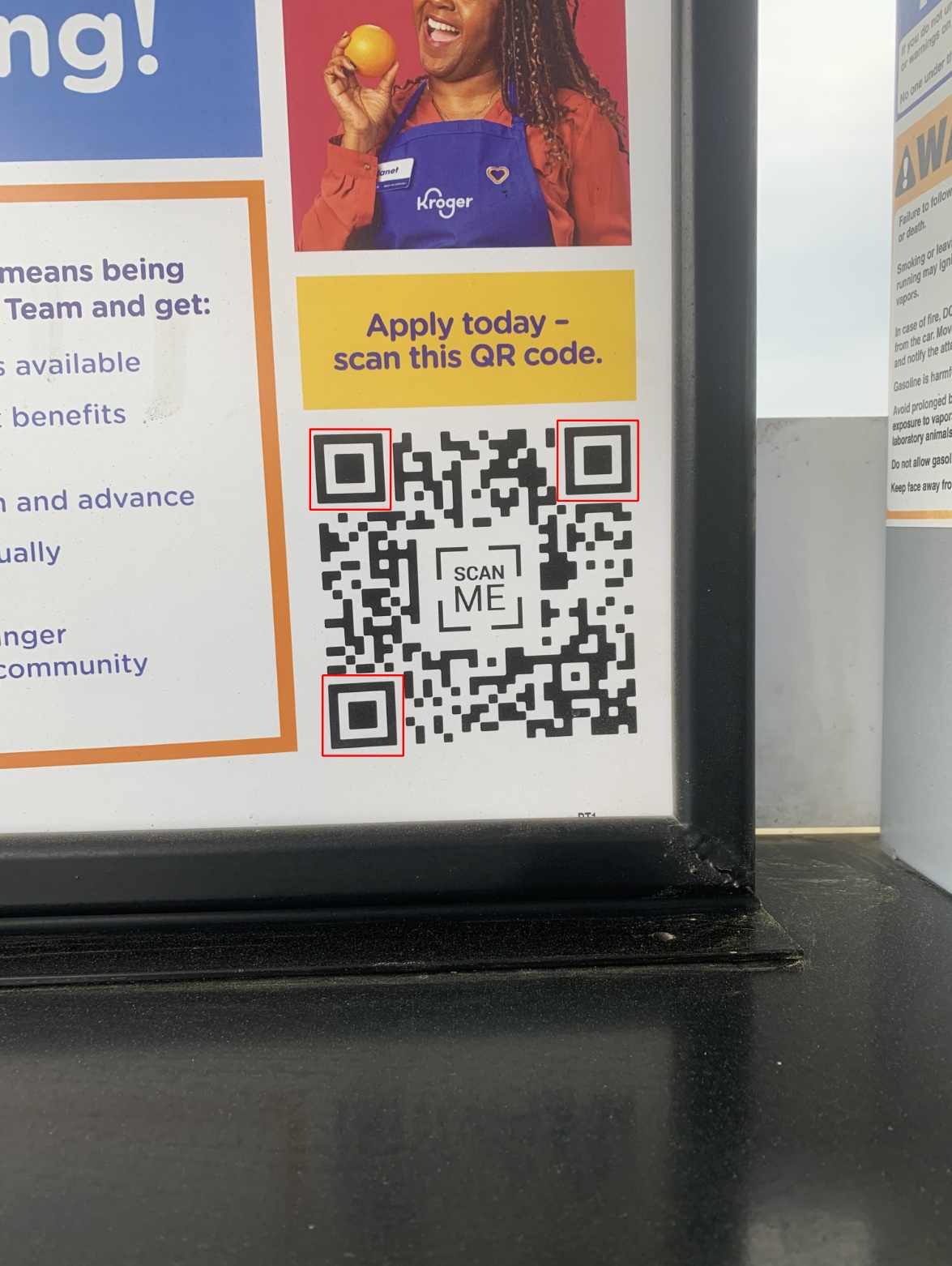
**Part 2d**

QR\_corners\_matched\_QR\_A.png

Qr code

Description automatically generated

QR\_corners\_matched\_QR\_B.png

****

QR\_corners\_matched\_QR\_C.png

****

QR\_corners\_matched\_QR\_D.png 

QR\_corners\_matched\_QR\_E.png



**Part 2g**

Affine\_Transformed\_QR\_A.png



Affine\_Transformed\_QR\_B.png



Affine\_Transformed\_QR\_C.png



Affine\_Transformed\_QR\_D.png



Affine\_Transformed\_QR\_E.png



**Part 3:**

The method for finding the QR corners used on the A-D did not work E. The method for finding the QR corners on the was simply changing the size of the template from a 10x10 to a 128x128 until 3 only templates were matched. All other template sizes gave multiple template matches and that is because the template was too large or too small. This might not be the absolute best method for finding the three corners, but with all of the other automation that organizes sorts the corners, the correct orientation was placed upright when using the affine method.

Code

Homework2.py:

"""

Created on Sat Jul 16 22:56:40 2022

@author: agarc

"""

import cv2

import numpy as np

import math

# Create a QR Template

qr\_template = np.zeros((128,128), np.uint8)

# Draw white border

cv2.rectangle(qr\_template

, pt1 = (0,0)

, pt2 = (128,128)

, color = (255,255,255)

, thickness = -1)

# Draw black border

cv2.rectangle(qr\_template

, pt1 = (12,12)

, pt2 = (116,116)

, color = (0,0,0)

, thickness = -1)

# Draw white box

cv2.rectangle(qr\_template

, pt1 = (24,24)

, pt2 = (102,102)

, color = (255,255,255)

, thickness = -1)

# Draw inner black box

cv2.rectangle(qr\_template

, pt1 = (42,42)

, pt2 = (84,84)

, color = (0,0,0)

, thickness = -1)

# Show and save image

cv2.imshow('QR Template',qr\_template)

cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite("qr\_template.png", qr\_template)

#%%

def get\_qr\_loc(qr\_img\_grey, qr\_template):

# Width and Height of template

w, h = qr\_template.shape[::-1]

# Run template matching

result = cv2.matchTemplate(qr\_img\_grey, qr\_template, cv2.TM\_CCOEFF\_NORMED)

# Part 2C Show Correlated Result

cv2.imshow(f'QR Matching {filename}', result)

cv2.waitKey(0)

cv2.imwrite(f"matchTemp\_reults\_{filename}", result\*255)

#

# Use Numpy Stuff to create 3-element list

#

# Find best matches

min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)

threshold = max\_val\*0.8

loc = np.where(result >= threshold)

# Iterate over loc to create list of (r,c,i)

loc\_list = [(r, c, result[r,c]) for r,c in zip(loc[0], loc[1])]

# sort by intensity

sorted\_loc = loc\_list.copy()

for i in range(0, len(sorted\_loc)):

for j in range(0, len(sorted\_loc)-i-1):

if (sorted\_loc[j][2] < sorted\_loc[j + 1][2]):

temp = sorted\_loc[j]

sorted\_loc[j]= sorted\_loc[j + 1]

sorted\_loc[j + 1]= temp

# Get points that are greater than template distance apart

#sorted\_loc\_final = []

for idx1 in range(len(sorted\_loc)):

for idx2 in range(len(sorted\_loc)):

if idx1 == idx2:

pass

elif type(sorted\_loc[idx1]) == int or type(sorted\_loc[idx2]) == int:

pass

else:

r1 = sorted\_loc[idx1][0]

c1 = sorted\_loc[idx1][1]

r2 = sorted\_loc[idx2][0]

c2 = sorted\_loc[idx2][1]

d = math.dist((r1,c1), (r2,c2))

if d < w:

sorted\_loc[idx2] = 0

# Remove all 0 from list

try:

while True:

sorted\_loc.remove(0)

except ValueError:

pass

return sorted\_loc

#%%

filenames = ['QR\_A.png', 'QR\_B.png', 'QR\_C.png', 'QR\_D.png', 'QR\_E.png']

for filename in filenames:

# Read in IMG as color and greyscale

qr\_img = cv2.imread(filename)

qr\_img\_grey = cv2.imread(filename,0)

sorted\_loc = get\_qr\_loc(qr\_img\_grey, qr\_template)

# Width and Height of template

w, h = qr\_template.shape[::-1]

# Is there more than 3 template matches?

if len(sorted\_loc) > 4 or len(sorted\_loc) < 3:

for scale in range(10,128):

new\_template = cv2.resize(qr\_template, [scale,scale])

sorted\_loc = get\_qr\_loc(qr\_img\_grey, new\_template)

print(f"Num Matched: {len(sorted\_loc)}, Scale: {scale}x{scale}")

if len(sorted\_loc) == 3:

break

# Organize into top\_left, bottom\_left, top\_right

sorted\_loc\_final = sorted\_loc.copy()

for i in range(0, len(sorted\_loc\_final)):

for j in range(0, len(sorted\_loc\_final)-i-1):

if (sorted\_loc\_final[j][0] > sorted\_loc\_final[j + 1][0]):

temp = sorted\_loc\_final[j]

sorted\_loc\_final[j]= sorted\_loc\_final[j + 1]

sorted\_loc\_final[j + 1] = temp

# Check if first two need to be swapped

if sorted\_loc\_final[0][1] > sorted\_loc\_final[1][1]:

temp = sorted\_loc\_final[0]

sorted\_loc\_final[0] = sorted\_loc\_final[1]

sorted\_loc\_final[j + 1] = temp

# Draw rectangles

for row\_col in sorted\_loc\_final:

r,c,\_ = row\_col

cv2.rectangle(qr\_img, (c, r), (c+w, r+h), (0,0,255), 2)

# Part 2b Print to console filename and location of 3 markers

print(f"Filename: {filename}")

for i in range(len(sorted\_loc\_final)):

print(f"(Row, Col, Intensity): {sorted\_loc\_final[i]}")

#

# Part 2d

#

# Show Image

cv2.imshow(f'QR\_corners\_matched\_{filename}', qr\_img)

cv2.waitKey(0)

cv2.imwrite(f'QR\_corners\_matched\_{filename}', qr\_img)

#

# Part 2e

#

img = qr\_img\_grey.copy()

img\_rows, img\_cols = img.shape

input\_pts = np.float32([[sorted\_loc\_final[0][1], sorted\_loc\_final[0][0]]

, [sorted\_loc\_final[1][1]+w, sorted\_loc\_final[1][0]]

, [sorted\_loc\_final[2][1], sorted\_loc\_final[2][0]+h]])

#

# Part 2f

# Apply the affine transformation using cv2.warpAffine()

#

output\_pts2 = np.float32([[50,50], [250,50], [50,250]])

M = cv2.getAffineTransform(input\_pts, output\_pts2)

warp\_dst\_rot = cv2.warpAffine(img, M, (300, 300))

# Display the image

cv2.imshow(f'Affine Image {filename}', warp\_dst\_rot)

cv2.waitKey(0)

cv2.imwrite(f'Affine\_Transformed\_{filename}', qr\_img)