Andrew Garcia
7/21/2022

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)

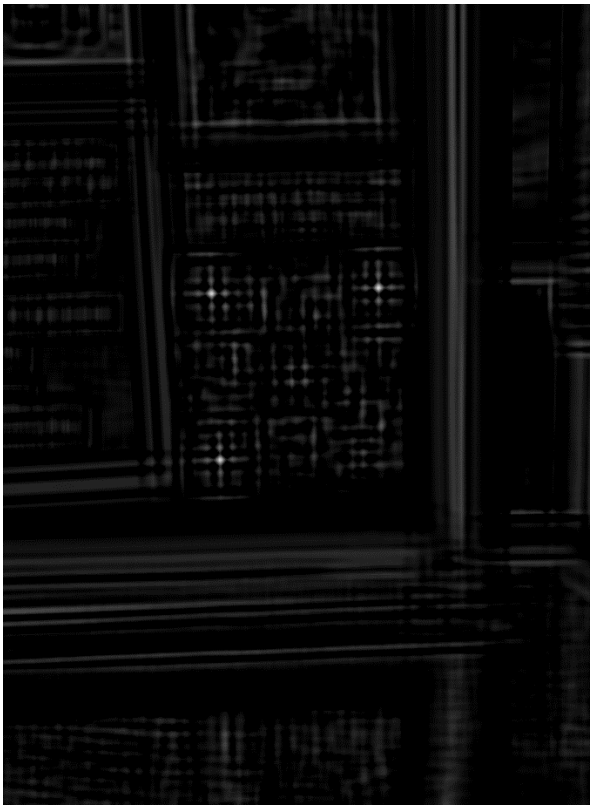
(Row, Col, Intensity): (580, 370, 0.48469767)

Andrew Garcia
7/21/2022

Part 2c:

matchTemp_reults_QR_A.png

matchTemp_reults_QR_B.png

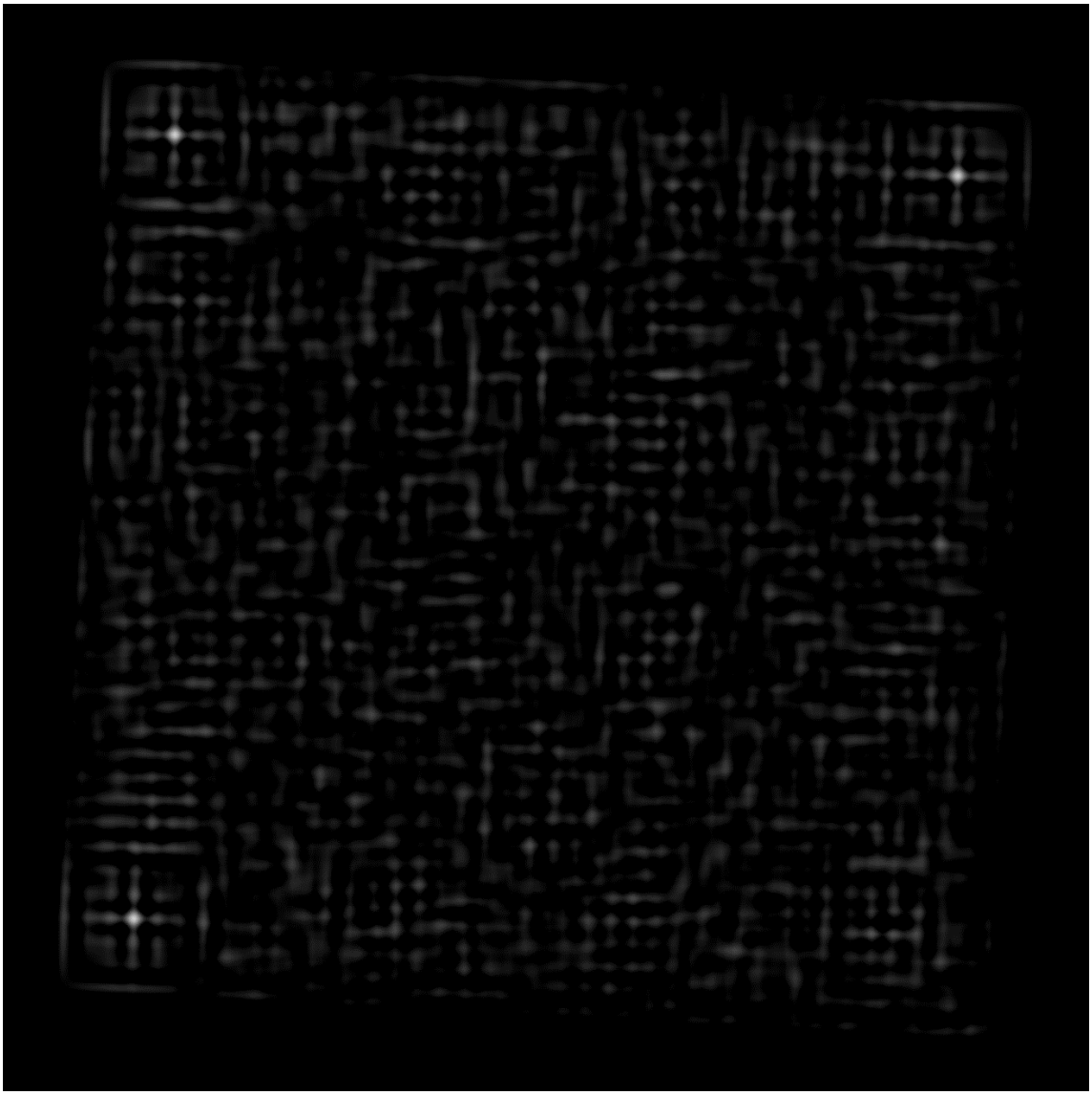


Andrew Garcia
7/21/2022

matchTemp_reults_QR_C.png

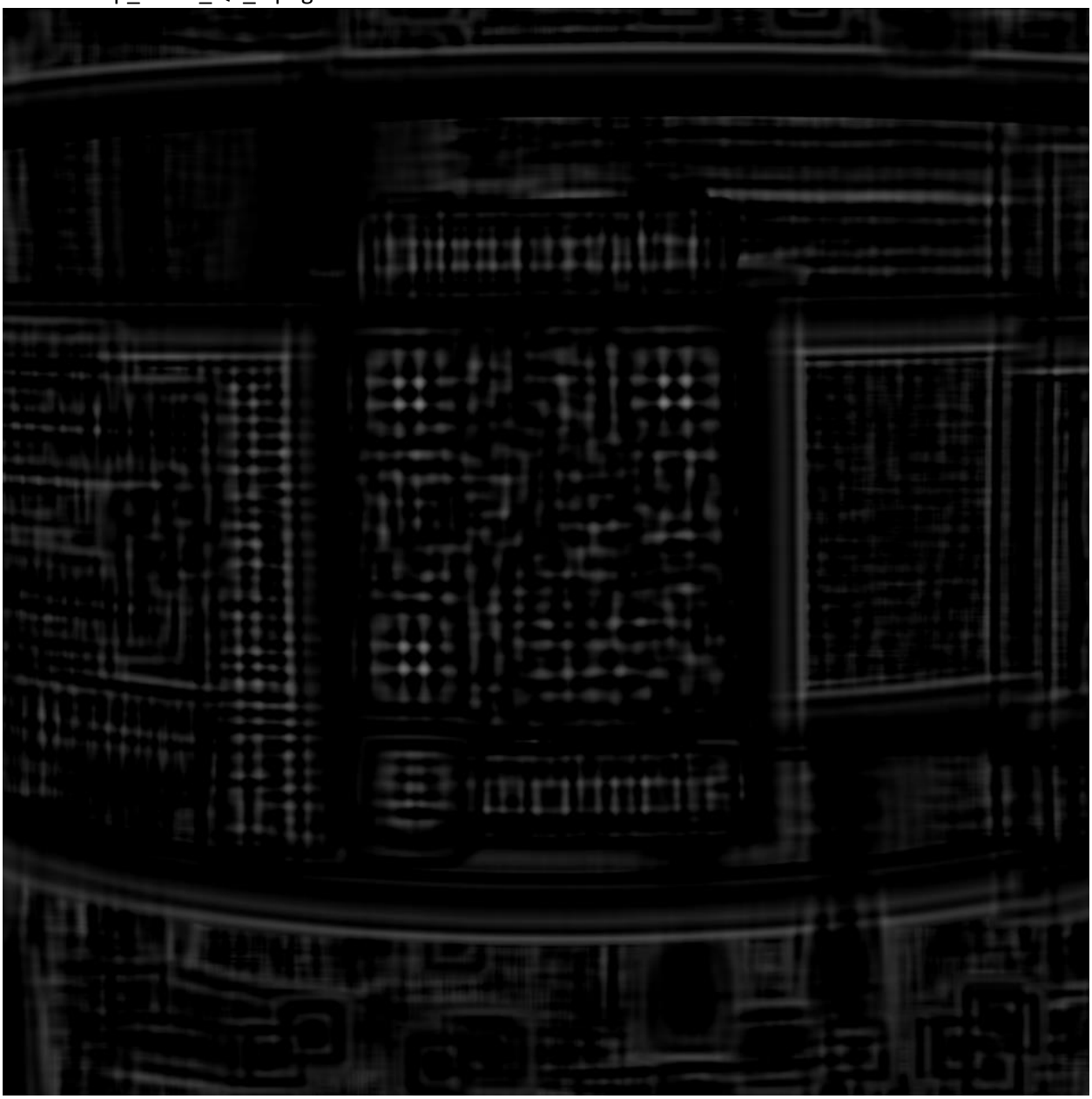
Andrew Garcia
7/21/2022

matchTemp_reults_QR_D.png



Andrew Garcia
7/21/2022

matchTemp_reults_QR_E.png



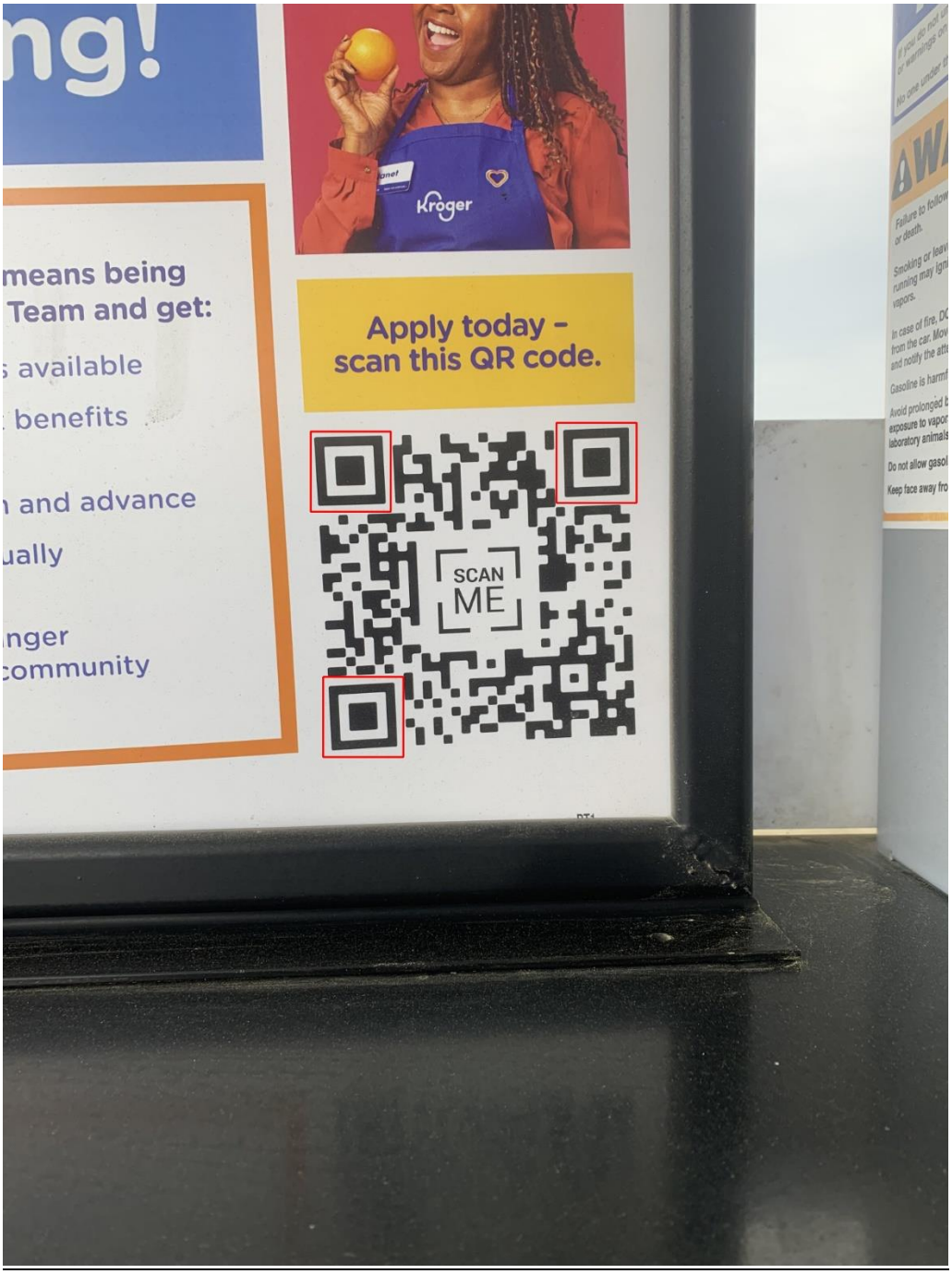
Andrew Garcia
7/21/2022

Part 2d

QR_corners_matched_QR_A.png

Andrew Garcia
7/21/2022

QR_corners_matched_QR_B.png



Andrew Garcia
7/21/2022

QR_corners_matched_QR_C.png

Andrew Garcia
7/21/2022

QR_corners_matched_QR_D.png

Andrew Garcia
7/21/2022

QR_corners_matched_QR_E.png

Andrew Garcia
7/21/2022

Part 2g

Affine_Transformed_QR_A.png

Affine_Transformed_QR_B.png

Andrew Garcia
7/21/2022

Affine_Transformed_QR_C.png

Affine_Transformed_QR_D.png

Andrew Garcia
7/21/2022

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.

Andrew Garcia
7/21/2022

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)
```