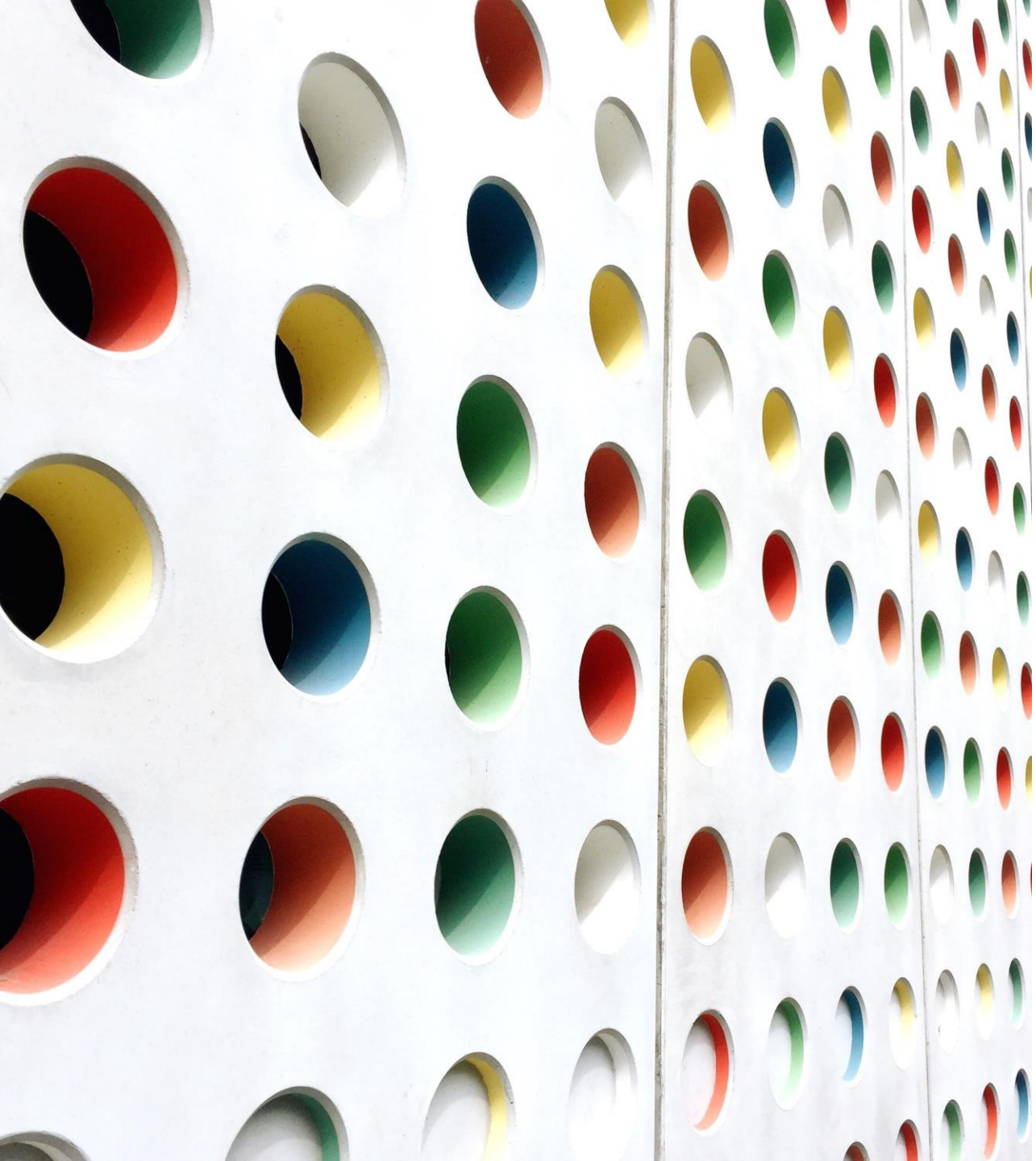


Computer Vision Homework 2

Matt Ferguson

July 20th, 2022

Requirements

- For a set of five QR codes images:
  - Develop a suitable QR positioning square template
  - Match the template to the three QR positioning squares in each image
  - Record the image file name and locations of the positioning squares
  - Record and display the template match map result
  - Place red rectangles about the matched QR positioning squares
  - Perform an affine transform to remap the marker locations and rescale the image to 300x300

Criteria
Create template for matching
Find and select the three marker locations
Print locations to console, and write annotated image to disk
Create proper affine transformation and transformed image; write to disk
Modify code to also work on image QR_E

Template Creation

```
w=128
x=16
p=(w-7*x)//2

template=np.zeros((x*7+2*p,x*7+2*p,3),np.uint8)

template=cv2.rectangle(template,(0,0),(w,w),(255,255,255),-1)
template=cv2.rectangle(template,(p,p),(w-p-1,w-p-1),(0,0,0),-1)
template=cv2.rectangle(template,(p+x,p+x),(w-p-1-x,w-p-1-x),(255,255,255),-1)
template=cv2.rectangle(template,(p+x+x,p+x+x),(w-p-1-x-x,w-p-1-x-x),(0,0,0),-1)

template_out=cv2.imwrite(r'C:\Users\Matt\Desktop\Virginia Tech\CV\template_AD.png', template)
```

We programmatically create two templates. One for QR's A-D and one for QR E. We invoke np.zeros to create our image array and cv2.rectangle() to draw nested squares that in total represent our QR positioning square. Our QR positioning square follows a 1:1:3:1:1 ratio. Our templates are 128x128 but note there is significant white padding around the squares. The difference in size of the template box widths is due to the difference of scale in the QR positioning squares as they appear in images A-D vs image E. Image E has a smaller positioning square width. Our templates are input into our template matching script and our script will take both templates for template matching and keep only the match map with the best template matching result.

Template Matching

We iterate through each provided image and each of the two templates. The two templates are used for matching on each of the images and the resulting match map with the highest average intensity is kept. The larger template matches best for A-D and the small template matches best for E.

We invoke gaussian blur on our template and image to reduce the effect of noise. Next, we use cv2.matchTemplate which passes our template over every pixel of the QR image and returns a resulting match map. We use normalized cross correlation. We remap our match map to 0-255 basis using min-max normalization.

We build a list ordered by intensity and take the 300 most intense pixel locations and pass them to a K means model (K=3).Our resulting 3 means locations are the centroids we use to draw rectangles around the QR positioning squares.

After ordering our centroids to appear in the same quadrant/order as our affine co-ordinates we create a warp matrix using the centroids and the provided co-ordinates to create a warp matrix and then apply an affine transformation to the original image.

Finally, we output the images of the match maps, the located centroids, and the affine transformed output.

```
import cv2
import numpy as np
from sklearn.cluster import KMeans

qrs=['A','B','C','D','E']
for qr in qrs:

    template_AD=cv2.imread(r'C:\Users\Matt\Desktop\Virginia Tech\CV\template_AD.png',0)
    template_E=cv2.imread(r'C:\Users\Matt\Desktop\Virginia Tech\CV\template_E.png',0)
    img = cv2.imread(r'C:\Users\Matt\Desktop\Virginia Tech\CV\QR_'+qr+'.png', 0)
    img_origin = cv2.imread(r'C:\Users\Matt\Desktop\Virginia Tech\CV\QR_'+qr+'.png')
    filename= r'QR_'+qr+'.png'

    img=img.astype(np.float32)
    template_AD=template_AD.astype(np.float32)
    template_E=template_E.astype(np.float32)

    # Blur masked image and template
    img=img.astype(np.float32)
    img=cv2.GaussianBlur(img,(15,15),0)
    template_AD=cv2.GaussianBlur(template_AD,(3,3),0)
    template_E=cv2.GaussianBlur(template_E,(3,3),0)

    res_AD=cv2.matchTemplate(img, template_AD, cv2.TM_CCORR_NORMED)
    res_E=cv2.matchTemplate(img, template_E, cv2.TM_CCORR_NORMED)
    min_val_AD, max_val_AD, min_loc_AD, max_loc_AD = cv2.minMaxLoc(res_AD)
    min_val_E, max_val_E, min_loc_E, max_loc_E = cv2.minMaxLoc(res_E)

    if max_val_AD>max_val_E:
        res=res_AD
    if max_val_AD<max_val_E:
        res=res_E

    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    res=255*(res-min_val)/(max_val-min_val)
    # res[res<50]=0

    h,w=img.shape[:2]
    cx,cy=w//2,h//2

    qr_max_arr=np.empty((3,(h-127)*(w-127)))
    counter=0
    for r in range(h-127):
        for c in range(w-127):
            if res[r,c]>200:
                qr_max_arr[:,counter] = [res[r,c], r, c]
                counter+=1
```

```
    qr_max_arr = np.transpose(qr_max_arr)
    qr_max_arr=qr_max_arr[~np.all(qr_max_arr == 0, axis=1)]
    qr_max_arr = qr_max_arr[qr_max_arr[:, 0].argsort()]
    qr_max_arr=qr_max_arr[-300:]

    # Kmeans clustering analysis of maxima
    kmean=KMeans(n_clusters=3)
    kfit=kmean.fit(X=qr_max_arr[:,1:])
    kpredict=kmean.predict(qr_max_arr[:,1:])

    centroids=(kmean.cluster_centers_)
    centroids=centroids.astype(int)
    img=cv2.cvtColor(img,cv2.COLOR_GRAY2RGB)
    cv2.rectangle(img,(centroids[0,1],centroids[0,0],),(centroids[0,1]+128,centroids[0,0]+128),(0,0,255),2)
    cv2.rectangle(img,(centroids[1,1],centroids[1,0]),(centroids[1,1]+128,centroids[1,0]+128),(0,0,255),2)
    cv2.rectangle(img,(centroids[2,1],centroids[2,0]),(centroids[2,1]+128,centroids[2,0]+128),(0,0,255),2)

    centroids=kmean.cluster_centers_
    xymean=centroids.mean(axis=0)
    xmean=xymean[1]
    ymean=xymean[0]

    for x in range(3):
        if centroids[x,1]<xmean and centroids[x,0]<ymean:
            a_x=centroids[x,1]+64
            a_y=centroids[x,0]+64
        elif centroids[x,1]>xmean and centroids[x,0]<ymean:
            b_x=centroids[x,1]+64
            b_y=centroids[x,0]+64
        elif centroids[x,1]<xmean and centroids[x,0]>ymean:
            c_x=centroids[x,1]+64
            c_y=centroids[x,0]+64

    A=50,50
    B=50,250
    C=250,50
    destinations=np.array([A,C,B]).astype(np.float32)
    destinations=np.ndarray.copy(destinations, order='C')
    source=np.array([[a_x,a_y],[b_x,b_y],[c_x,c_y]])
    source=np.ndarray.copy(source, order='C')
    source=source.astype(np.float32)
    warp_mat=cv2.getAffineTransform(source, destinations)
    warped=cv2.warpAffine(img_origin,warp_mat,(300,300))


    print(centroids)
    img_out=cv2.imwrite(r'C:\Users\Matt\Desktop\Virginia Tech\CV\QR\Mask\qr_'+qr+'_res.png', res)
    mimg_out=cv2.imwrite(r'C:\Users\Matt\Desktop\Virginia Tech\CV\QR\Mask\qr_'+qr+'rect.png', img)
    warp_out=cv2.imwrite(r'C:\Users\Matt\Desktop\Virginia Tech\CV\QR\Mask\warp'+qr+'.png', warped)
```

Results

We will display our results in the follow slides in a four-block template. The console prints of the centroids will be displayed on the left-hand side of the four-blocker. Note that we affine transform using the co-ordinates of the centers of the QR squares rather than the upper left-hand corner

The upper left image is the raw match map, where high intensity regions represent high correlation with the template. The bottom left image is a visualization of the input to the K-means model. The upper right image locates the centroids with red rectangles. The lower right image is the output of the warp affine transformation.

The black border around the images is for aesthetic purposes only and is not part of the raw output.

Results for each input image:

- Printed centroids and name of file
- Image of match map result
- Image of K-means input map
- Image of centroid locations
- Image of affine transformation

QR A

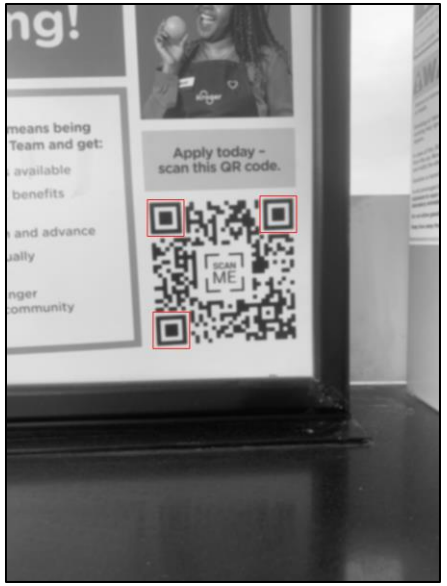
Printed QR A Centroids:

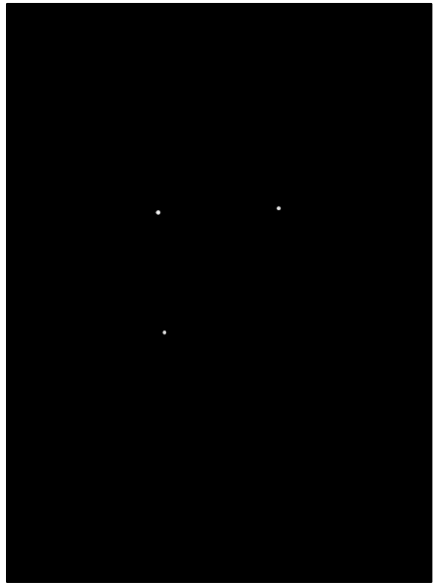
QR_A.png
[[526.49 , 39.18]
[291.55 , 261.86]
[297.5 , 33.12]]

QR B

Printed QR B Centroids:

QR_B.png
[[668.5 , 885.83]
[682. , 491.40]
[1074.82 , 511.95]]

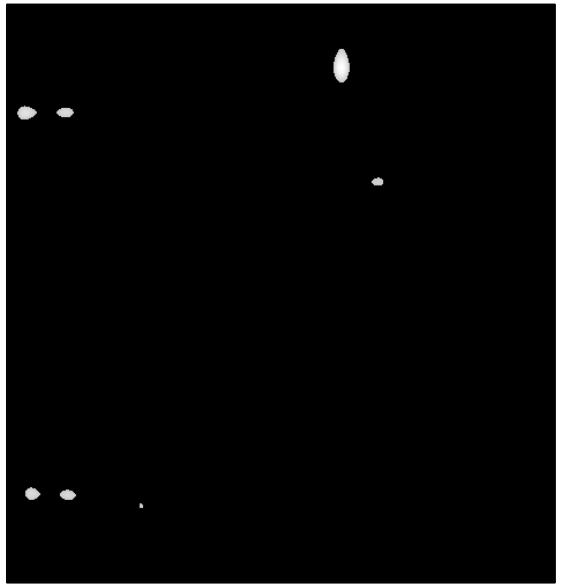




QR C

Printed QR C Centroids:

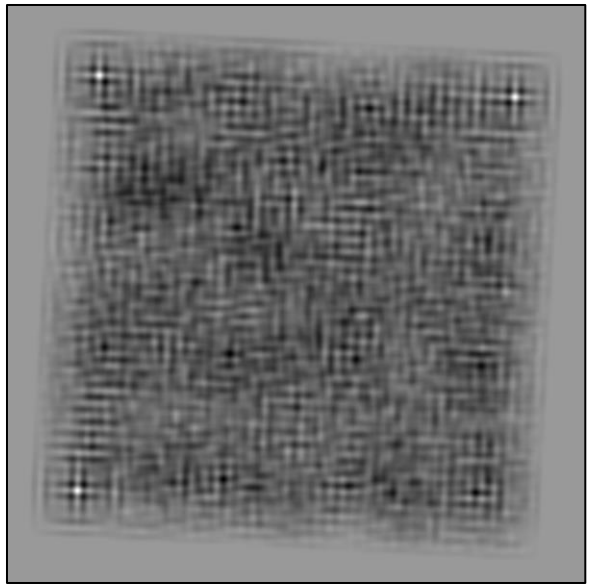
QR_C.png
[[52.94 , 288.24]
 [92.90 ,   17.48]
 [422.   ,   19.80]]

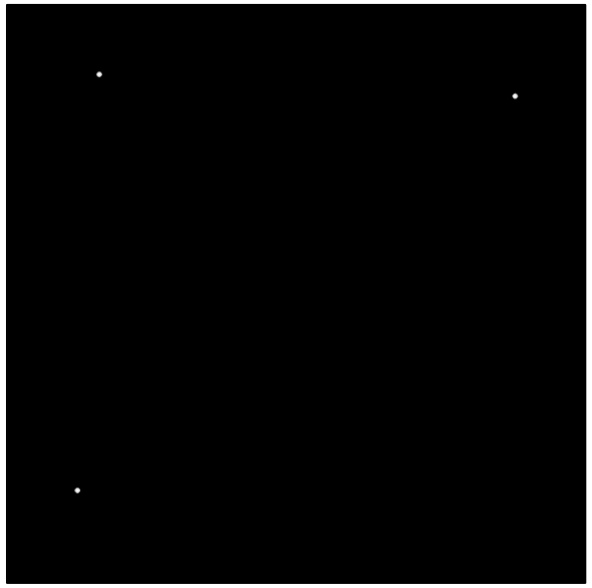


QR D

Printed QR D Centroids:

QR_D.png
[[228.94 , 1276.52]
[174.12 , 229.5]
[1221.06 , 174.52]]





QR E

Printed QR E Centroids:

QR_E.png
[[318.59 , 573.33]
 [553.14 , 342.76]
 [320.31 , 339.   ]]