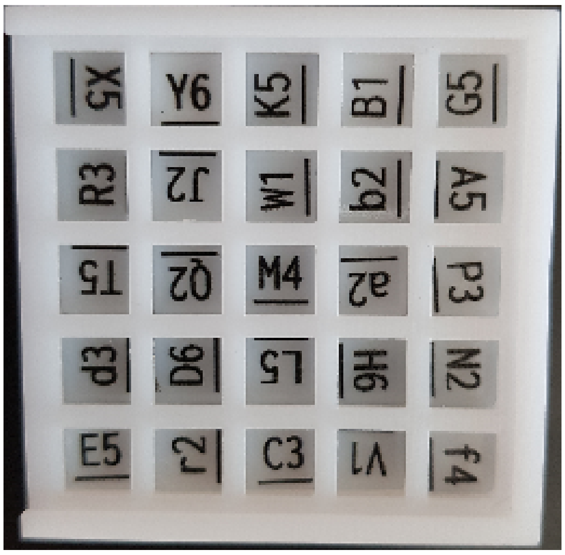
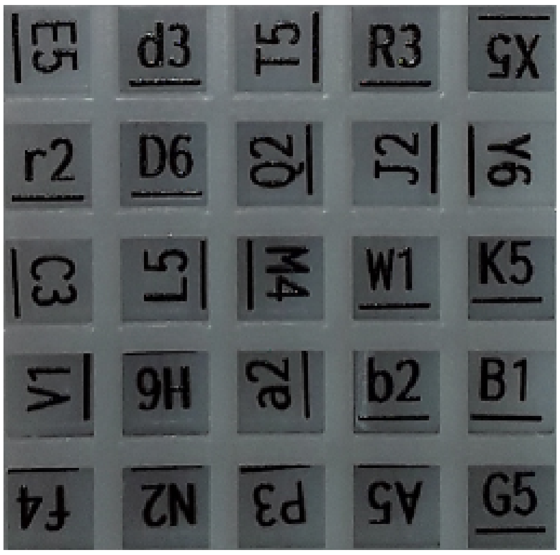
Square detection in image

Asked 5 years, 9 months ago   Modified 2 years, 8 months ago   Viewed 45k times

37

I am trying to detect all the squared shaped dice images so that i can crop them individually and use that for OCR. Below is the Original image:

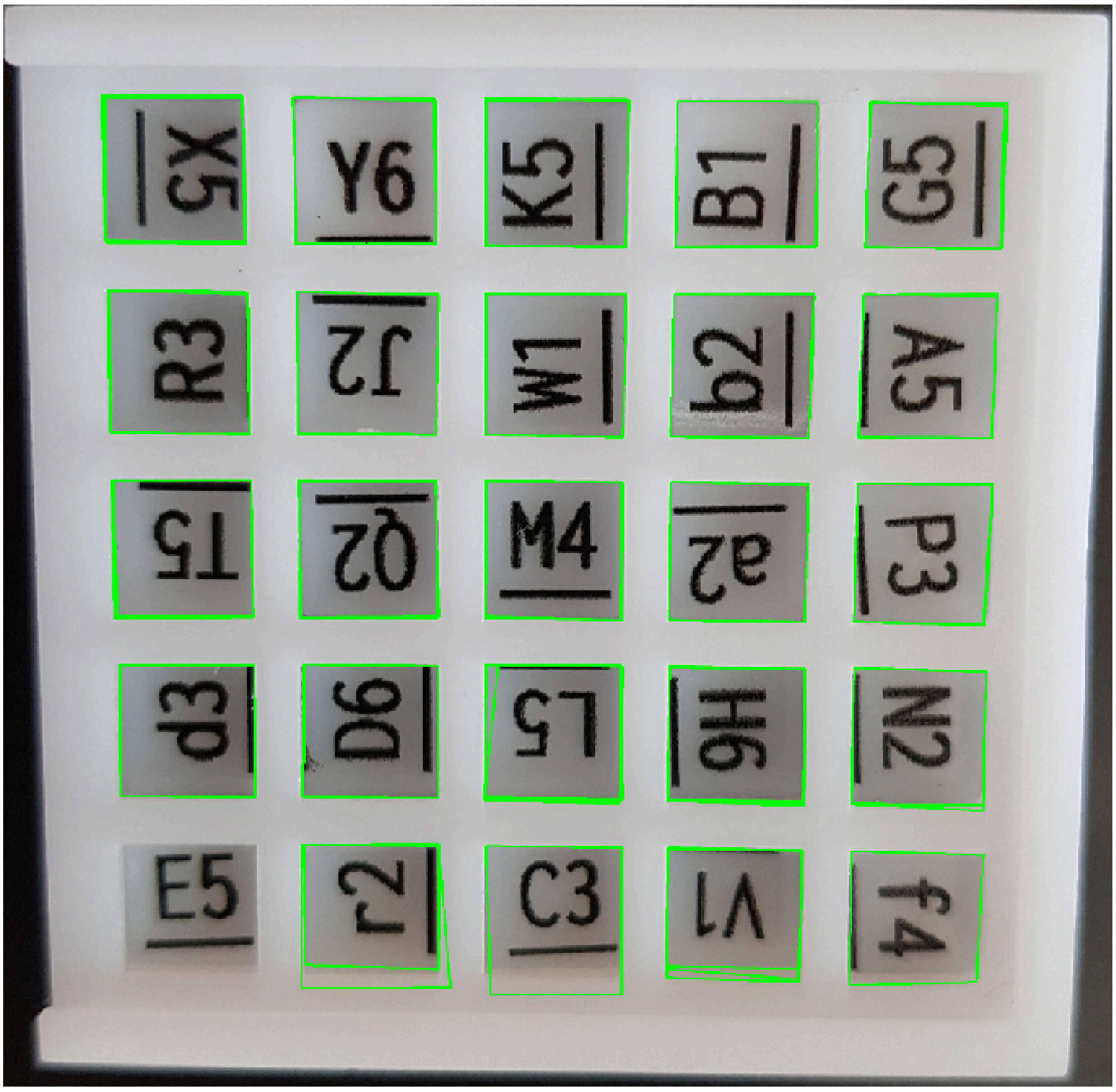



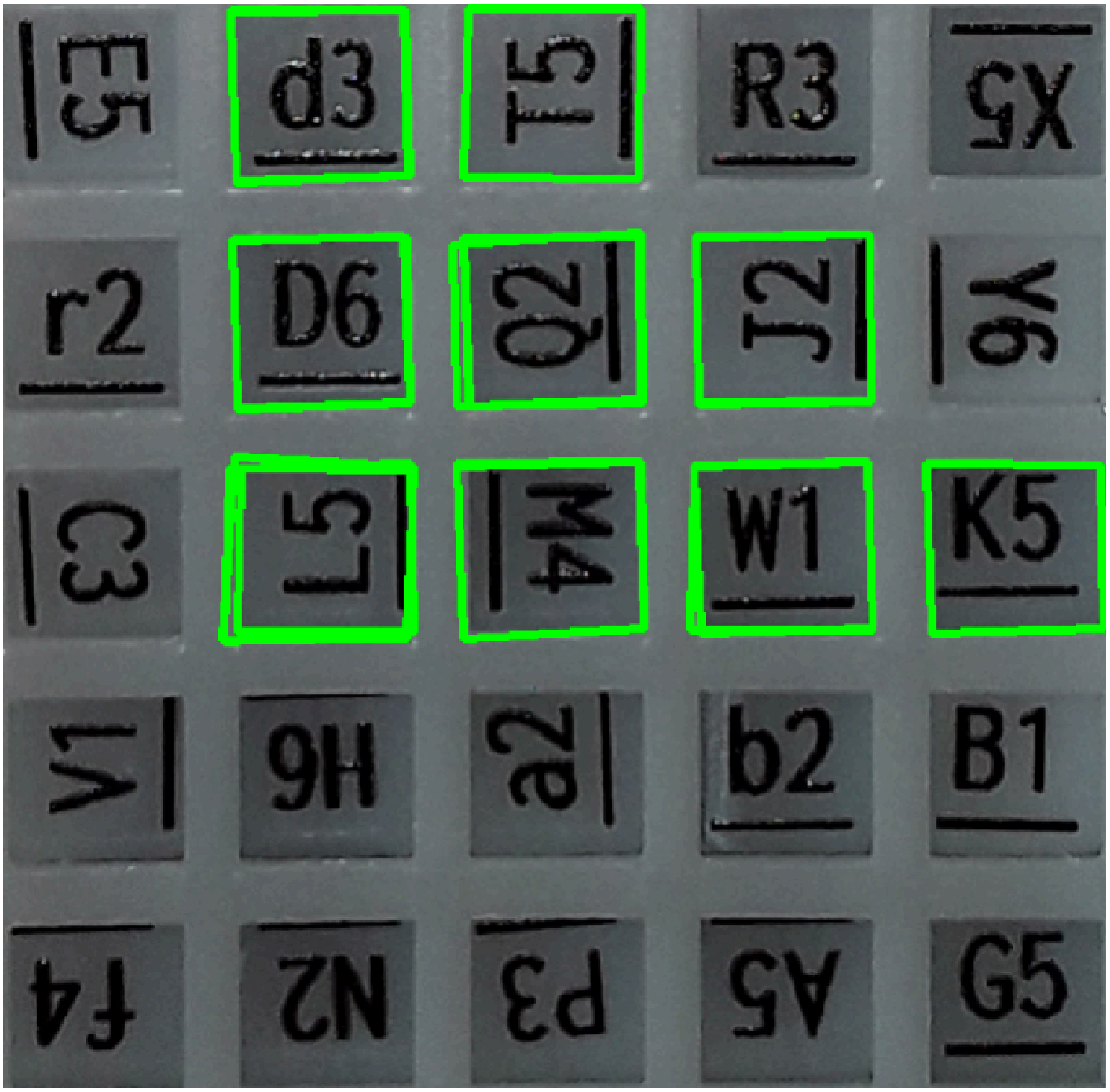
Here is the code i have got but it is missing some squares.

```
def find_squares(img):
    img = cv2.GaussianBlur(img, (5, 5), 0)
    squares = []
    for gray in cv2.split(img):
        for thrs in range(0, 255, 26):
            if thrs == 0:
                bin = cv2.Canny(gray, 0, 50, apertureSize=5)
                bin = cv2.dilate(bin, None)
            else:
                _retval, bin = cv2.threshold(gray, thrs, 255,
cv2.THRESH_BINARY)
            bin, contours, _hierarchy = cv2.findContours(bin, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
            for cnt in contours:
                cnt_len = cv2.arcLength(cnt, True)
                cnt = cv2.approxPolyDP(cnt, 0.02*cnt_len, True)
                if len(cnt) == 4 and cv2.contourArea(cnt) > 1000 and
cv2.isContourConvex(cnt):
                    cnt = cnt.reshape(-1, 2)
                    max_cos = np.max([angle_cos( cnt[i], cnt[(i+1) % 4],
cnt[(i+2) % 4] ) for i in range(4)])
                    #print(cnt)
                    a = (cnt[1][1] - cnt[0][1])

                    if max_cos < 0.1 and a < img.shape[0]*0.8:

                        squares.append(cnt)
    return squares

dice = cv2.imread('img1.png')
```

```
squares = find_squares(dice)
cv2.drawContours(dice, squares, -1, (0, 255, 0), 3)
```

Here are the Output images:





As per my analysis, some squares are missing due to missing canny edges along the dice because of smooth intensity transition between dice and background.

Given the constraint that there will always be 25 dices in square grid pattern (5*5) can we predict the missing square positions based on recognised squares? Or can we modify above algorithm for square detection algorithm?

python image opencv image-processing computer-vision

Share
Improve this question
Follow

edited Jul 25, 2019 at 1:48
fmw42
52.9k ● 10 ● 74 ● 89

asked Mar 14, 2019 at 18:26
flamelite
2,844 ● 4 ● 24 ● 42

Please include your analysis of *how* your program is failing to find other squares. Is it having problems identifying some edges? Does the image boundary confuse it? Does it not connect adjacent edges in some cases? We do expect you to provide a reasonable debugging pass;

simply showing us that "it doesn't work" is not a problem specification. – Prune Mar 14, 2019 at 18:50

@Prune thank for the suggestion, i have added my analysis about the challenge. – flamelite Mar 14, 2019 at 19:47

Check this out and help stackoverflow.com/q/77386078/19874226 – DsPro Oct 30, 2023 at 11:56

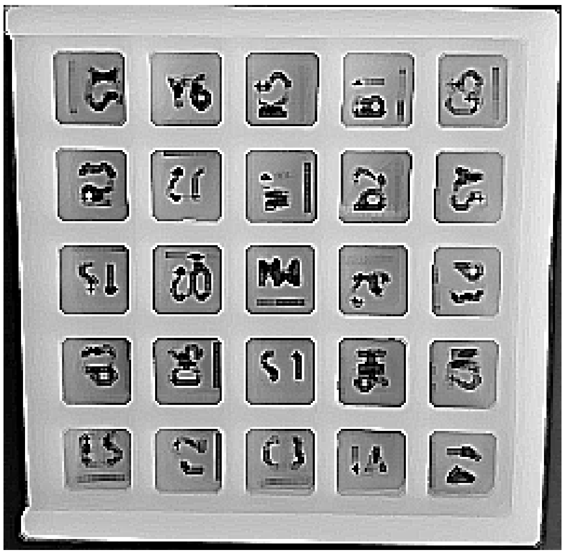
2 Answers

Sorted by: Highest score (default)

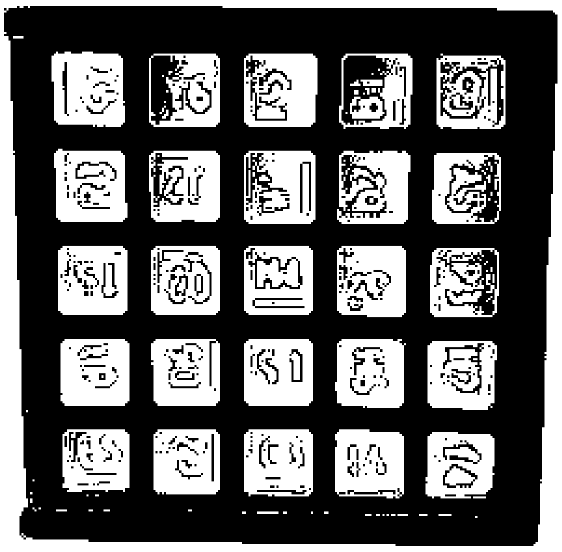
67

1. **Sharpen square edges.** Load the image, convert to grayscale, median blur to smooth, and sharpen to enhance edges.
2. **Obtain binary image and remove noise.** We threshold to obtain a black/white binary image. Depending on the image, Otsu's thresholding or adaptive thresholding would work. From here we create a rectangular kernel and perform morphological transformations to remove noise and enhance the square contours.
3. **Detect and extract squares.** Next we find contours and filter using minimum/maximum threshold area. Any contours that pass our filter will be our squares so to extract each ROI, we obtain the bounding rectangle coordinates, crop using Numpy slicing, and save each square image.

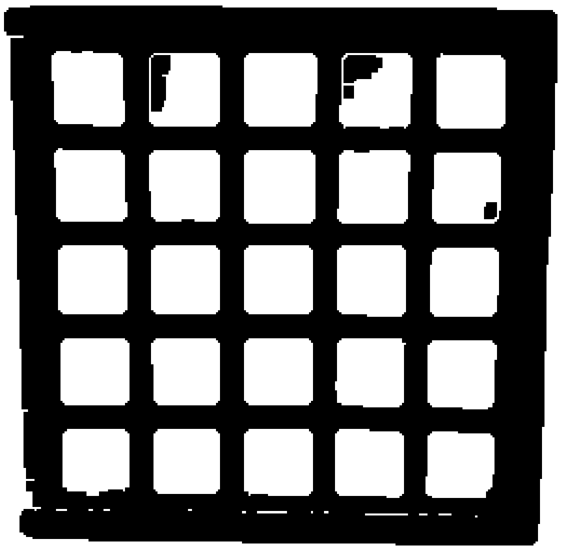
Sharpen image with `cv2.filter2D()` using a generic sharpening kernel, other kernels can be found here.



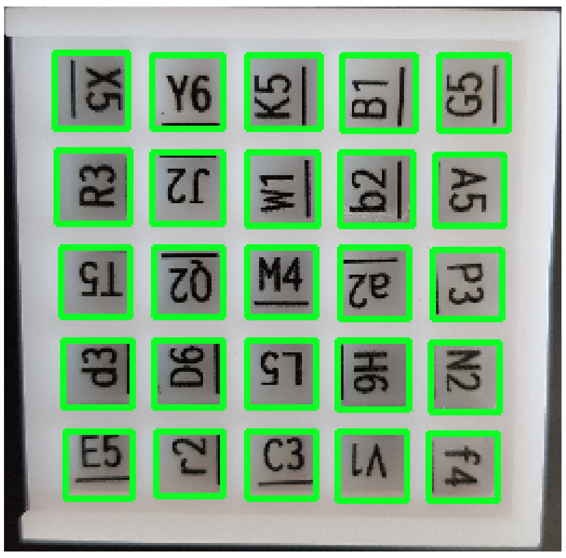
Now threshold to get a binary image



There's little particles of noise so to remove them, we perform morphological operations

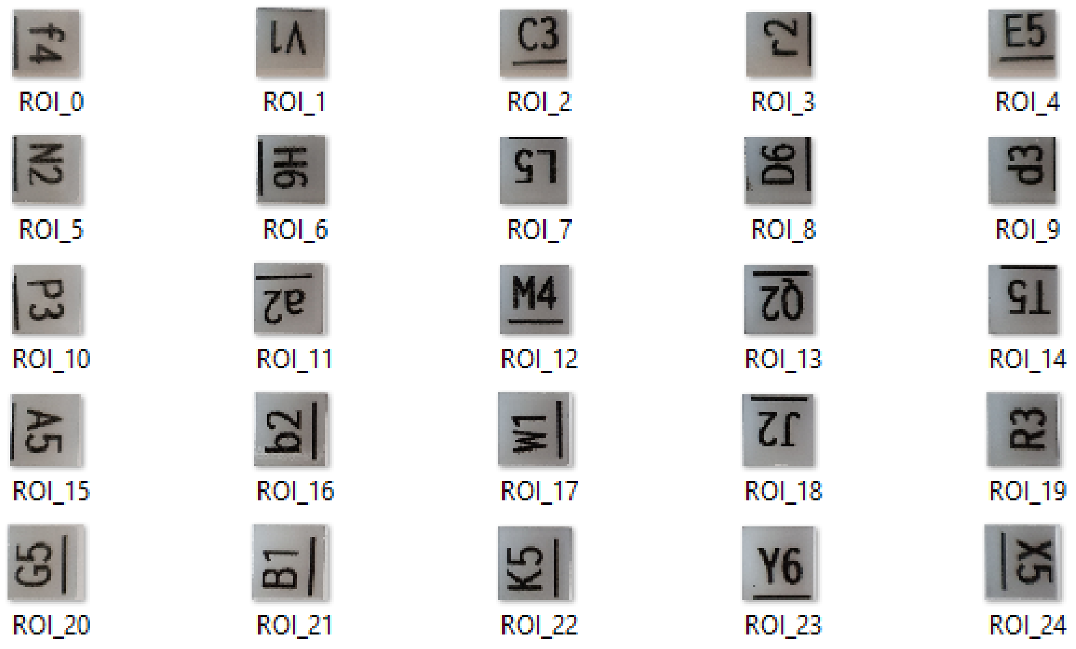


Next find contours and filter using `cv2.contourArea()` with minimum/maximum threshold values.



We can crop each desired square region using Numpy slicing and save each ROI like this

```
x,y,w,h = cv2.boundingRect(c)
ROI = image[y:y+h, x:x+w]
cv2.imwrite('ROI_{}.png'.format(image_number), ROI)
```



```
import cv2
import numpy as np

# Load image, grayscale, median blur, sharpen image
image = cv2.imread('1.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blur = cv2.medianBlur(gray, 5)
sharpen_kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
sharpen = cv2.filter2D(blur, -1, sharpen_kernel)

# Threshold and morph close
thresh = cv2.threshold(sharpen, 160, 255, cv2.THRESH_BINARY_INV)[1]
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
close = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=2)

# Find contours and filter using threshold area
cnts = cv2.findContours(close, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

min_area = 100
max_area = 1500
image_number = 0
for c in cnts:
    area = cv2.contourArea(c)
    if area > min_area and area < max_area:
        x,y,w,h = cv2.boundingRect(c)
        ROI = image[y:y+h, x:x+w]
        cv2.imwrite('ROI_{}.png'.format(image_number), ROI)
        cv2.rectangle(image, (x, y), (x + w, y + h), (36,255,12), 2)
        image_number += 1
```

```
cv2.imshow('sharpen', sharpen)
cv2.imshow('close', close)
cv2.imshow('thresh', thresh)
cv2.imshow('image', image)
cv2.waitKey()
```

Share
Improve this answer
Follow

edited Apr 8, 2022 at 10:08

answered Jul 25, 2019 at 1:36
nathancy
46.4k ● 15 ● 132 ● 153

4 thank you for great answer. I think line in loop `ROI = image[y:y+h, x:x+h]` should be `ROI = image[y:y+h, x:x+w]` ? – Dgan Sep 19, 2021 at 11:44

@Ganesh_Devlekar, yes you're right. Thanks. I updated it – nathancy Sep 20, 2021 at 9:48

4 wow,, this is astonishing – Yasir Hantoush Jun 16, 2022 at 15:42

2 This is an insanely in-depth and amazing answer. Bravo, sir. – Sharpienero Dec 22, 2022 at 3:35

3 This is not an answer, it's a complete tutorial. Nice! – Cagy79 Jul 22, 2023 at 20:55

0

That extra piece of information is absolutely golden. Yes, given the 5x5 matrix of dice, you can nail the positions quite well. The dice you *can* identify give you the center, size, and orientation of the dice. Simply continue those patterns along both axes. For your second pass, increase the contrast in each "region of interest" where you expect to find the edge of a douse (never say die!). You know within a few pixels where the edges will be: simply attenuate the image until you identify those edges.

Share Improve this answer Follow

answered Mar 14, 2019 at 21:02
Prune
77.8k ● 14 ● 62 ● 83