“Computer Vision”

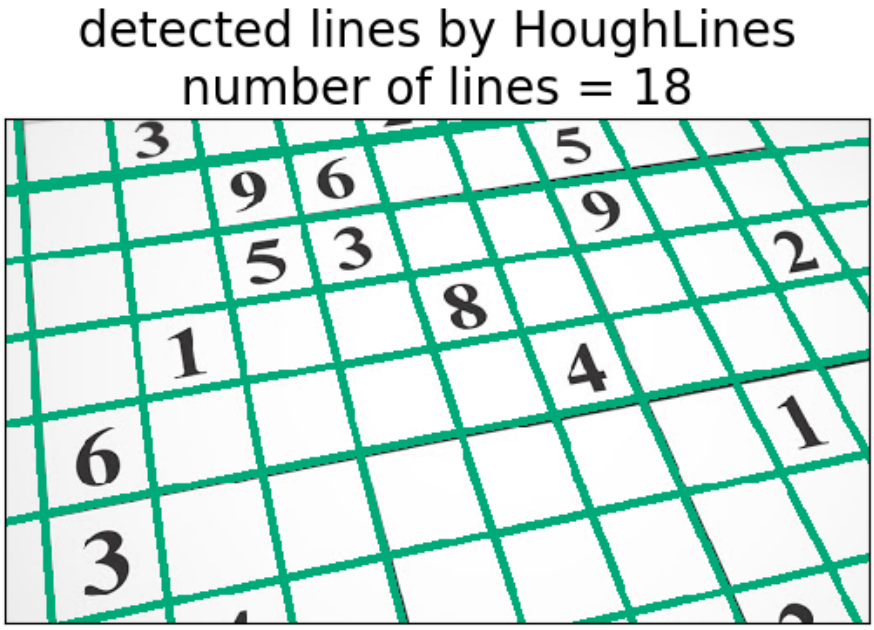
Shervin Halat

98131018

Homework 2

1.

Detected lines by HoughLines are shown with green lines:



As we can see in the figure above, all lines are perfectly detected with only one green line without any overlap by the help of following implemented function of ‘opt’ in related script.

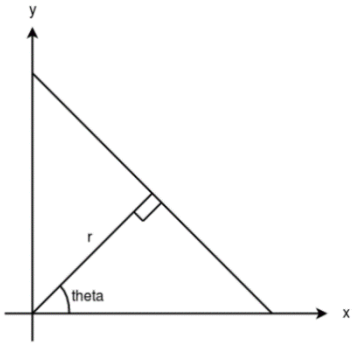
```
def opt(houghlines):
    for ind1 in range(len(lines)):
        temp = []
        for ind2 in range(len(lines)):
            if list(abs(lines[ind1]-lines[ind2])) < [10,1.5]:
                temp.append(lines[ind2])
        if len(temp)>1:
            temp.sort(key=lambda x: x[1])
            new = temp[len(temp)//2]
            lines[ind1] = new
    return lines
```

Explanation:

To solve this problem, first GaussianBlur function was applied with kernel of (3,3), then edges were detected using Canny function with 80 and 100 threshold levels; afterward, lines were extracted using HoughLines function, then in order to overcome overlapping lines, implemented ‘opt’ function was applied. ‘Opt’ function removes lines with very close properties; that is, for each line it compares both of its ‘rho’ and ‘theta’ with other lines and stores its close lines. Lastly, for each set of close lines, only the line with median ‘theta’ will be kept.

Parameters of HoughLines function:

a) First parameter is the binary input image after edge detection which we aim to find its lines by Hough operation.
b) Second parameter (rho, r) is the distance resolution of the accumulator matrix in pixels, considering the following plot. (“r” is vertical distance from the origin to corresponding line)



c) Third parameter (theta) is the angle resolution of the accumulator matrix in radians, considering the plot above. (“theta” is the angle between perpendicular line from the origin to a line of the image and the ‘x’ axis)

d) Fourth parameter (threshold) is a threshold by which greater values in the accumulator matrix are chosen and considered as parameters of detected lines. In other words, this threshold is the minimum vote each cell of accumulator should get to be considered as a line.
e) Fifth parameter (srn) is used as a parameter for an enhanced version of hough transform which detect lines more accurately. This parameter controls accumulator distance resolution so that the coarse accumulator distance resolution will be rho and the accurate accumulator resolution is rho/srn .
f) Sixth parameter (stn) works just like srn except it is used for the accumulator angle resolution theta.

2.

Detected circles by HoughCircles are shown with red lines:

As we can see in the figure above, only circle shaped coins are perfectly detected as circles by playing with the HoughCircles function parameters.

Explanation:

To solve this problem, first GaussianBlur function was applied with kernel of (3,3), then HoughCircles function was applied in order to detect circles with the following parameters:

```
detected_circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, 1, 10\
                                    , param1 = 120, param2 =120, minRadius = 10, maxRadius = 110)
```

It should be noted that lines are detected internally by HoughCircles function.

Parameters of HoughCircles function:

a) First parameter is the grayscale image in which we are to detect circles.
b) Second parameter is the method of detection that function will be used. Currently the only available method in Opencv is HOUGH_GRADIENT which uses gradient of the edges whereby the 3D accumulator matrix is shrunk to 2D matrix; therefore, the algorithm speeds up and needs less memory. That is, instead of using following equation,

$$(x_c - x_1)^2 + (y_c - y_1)^2 = r^2$$

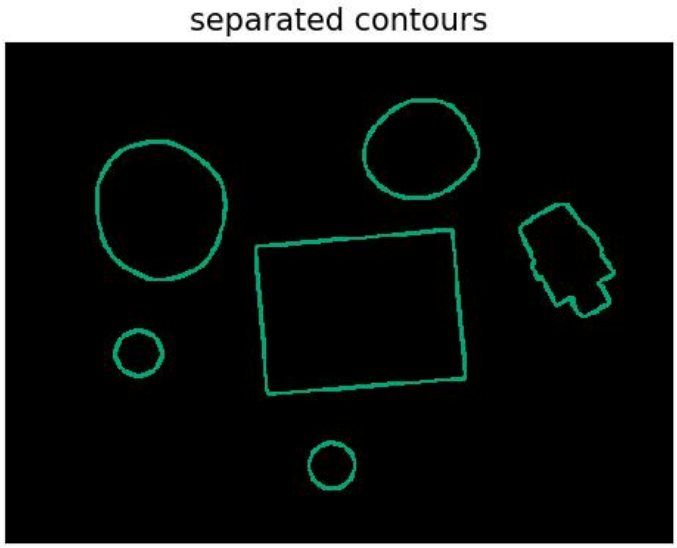
the following will be used:

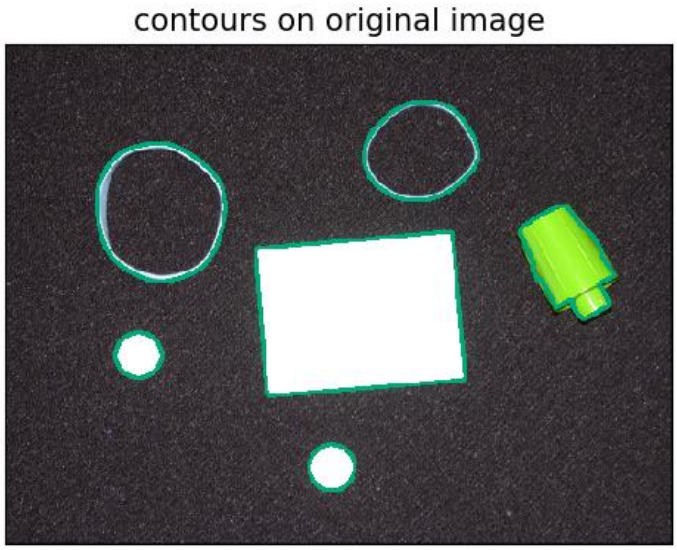
$$y_c = (y_1 - x_1 \tan\varphi) + x_c \tan\varphi$$

c) The third parameter (dp) is the ratio of the resolution of original image to the resolution of accumulator matrix. For example, if dp=1, the accumulator has the same resolution as the input image. If dp=2, the accumulator has half as big width and height.
d) The fourth parameter (minDist) controls the minimum distance between detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
e) The fifth parameter (param1) is the higher threshold for the canny edge detector function which is applied on the input image internally by HoughCircle function. It should be noted that the lower threshold for canny function is twice smaller than the given parameter.

f) The sixth parameter (param2) is the accumulator matrix threshold for the candidate detected circles. By increasing this value, we can ensure that less false circles are detected and vice versa.
g) The seventh parameter (minRadius) defines minimum circle radius to be detected.
h) The eighth parameter (maxRadius) defines maximum circle radius to be detected.

3.





Explanation:

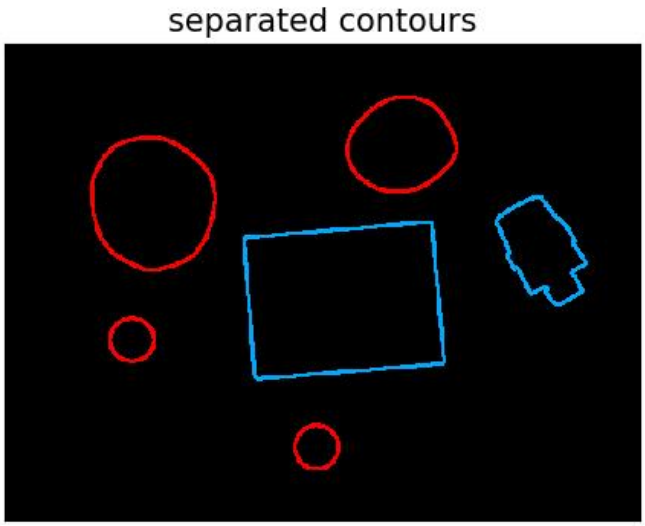
To solve this problem, first, the input image was blurred with gaussian kernel of size (5,5), then threshold function was applied with level of 108 in order to convert the image to binary. Lastly, with the help of findContours with the following parameters, active contours were detected.

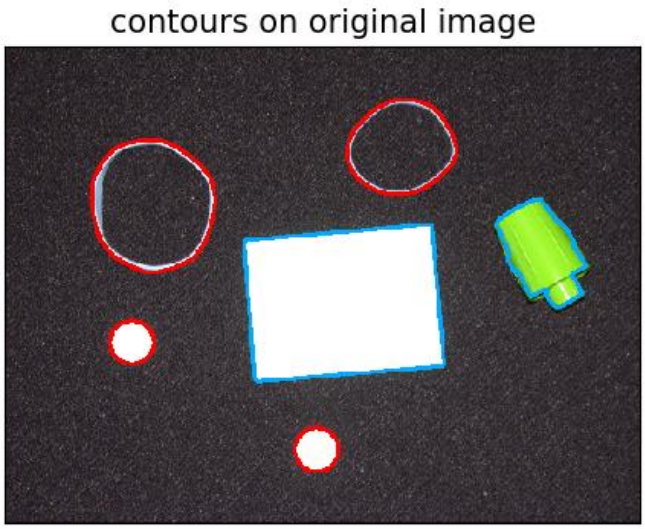
```
1 contours, hierarchy = cv2.findContours(thresh,  cv2.RETR_EXTERNAL  , cv2.CHAIN_APPROX_NONE)
```

Parameters of findContours function:

a) The first parameter is the binary input image which is the detected edges of the main image (e.g. by canny operation) or the binary threshold of main image.
b) The second parameter (mode) is contour retrieval mode. This parameter determines two different types of outputs. One type (which is specified by RETR_EXTERNAL) retrieves only the extreme outer contours and the other (e.g. RETR_LIST) retrieves all of the contours without establishing any hierarchical relationships. There are three other modes which differ from each other only by established hierarchical relationships.
c) The third parameter (method) is contour approximation method. In fact contours are a set of (x,y) coordinates that defines a shape boundary. The number of essential coordinates to be stored is defined by this parameter. For example, a line can be defined by only two points and there is no need for storing all points of a line. The argument CHAIN_APPROX_NONE stores all boundary points as against CHAIN_APPROX_SIMPLE which only stores end points of lines. In the following differences are shown:

4.





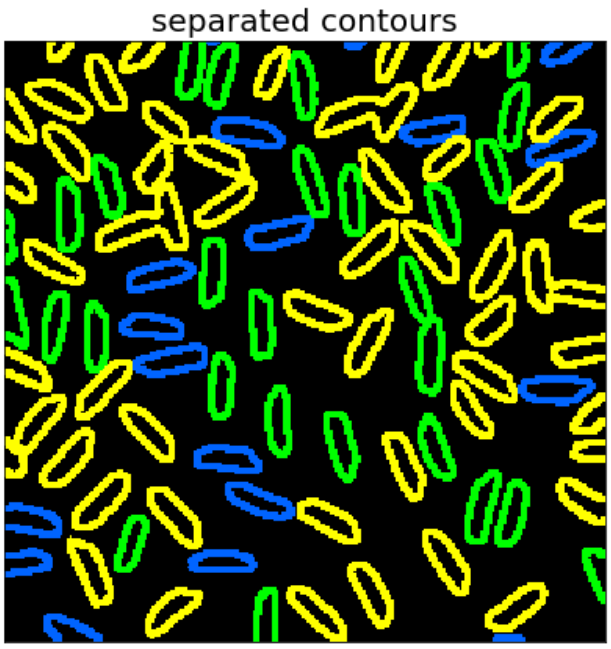
Implemented script to distinguish circles from other shapes:

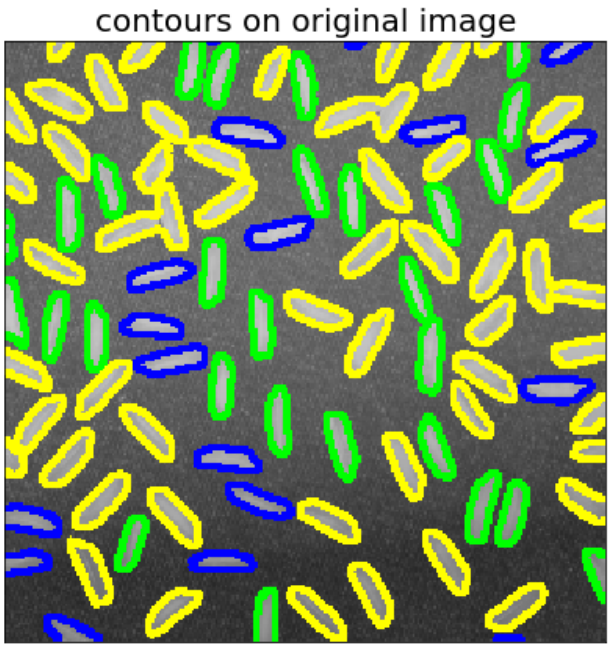
```
circles = []
others = []
for cont in contours:
    if abs(np.pi*(cv2.arcLength(cont,True)/(2*np.pi))**2 - cv2.contourArea(cont)) <= 1100:
        circles.append(cont)

for cont in contours:
    if cont not in circles:
        others.append(cont)
```

Explanation:

First, by considering each shape as a circle, with the help of arclength function, approximate radius of the shape is computed; then by this radius, an estimation of the expected area of the shape is computed. Afterward, by comparing the real area (which is obtained by contourArea function) with the estimated area, if the difference is lower than a specific level (in this case, 1100 was chosen by trial and error) the shape is considered as a circle and non-circle otherwise.

5.





Implemented script to distinguish horizontal, vertical, and others from each other:

```
verticals = []
horizontals = []
none = []
for cont in contours:
    x,y,width, height = cv2.boundingRect(cont)
    if width / height >= 1.8:
        horizontals.append(cont)
    elif height / width >= 1.7:
        verticals.append(cont)
    else:
        none.append(cont)
```

Explanation:

To differentiate between pieces of rice by their angles, the width and height of bounding rectangle of each piece was obtained by boundingRect function. Then, by computing ratio of width to height (or height to width) and comparing with a threshold (in this case 1.8 which was chosen by trial and error), all pieces were classified.