Dices Detection and Dots Counting

Petru-Alexandru Rotaru
SUN Yimeng
Hanna Smach
Ahmed Kallel

Presentation plan

- The Objective
- The Dataset
- Dice segmentation
- Dot segmentation and counting

Objective:

- The aim of the project is to extract the total number of points shown on an overhead picture of a group of dices.

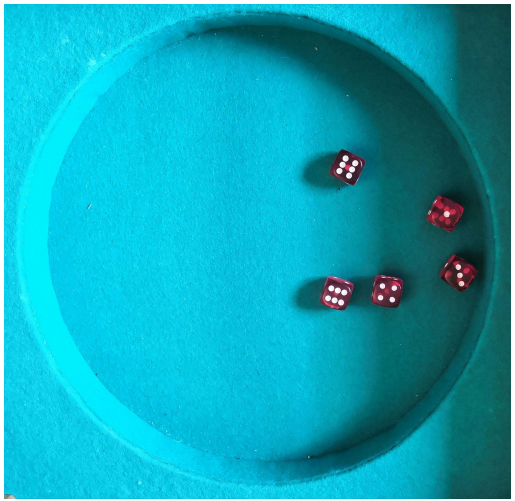
Assumptions:

- All the pictures have a uniform blue felt background.
- The dice are of similar size and have the same color.
- The pictures are taking from an overhead angle.

Dataset

- The dataset consists of 9 images
- All images contain 5 red dice on a light blue felt background
- Some of the images contain “obstacles”

Image examples

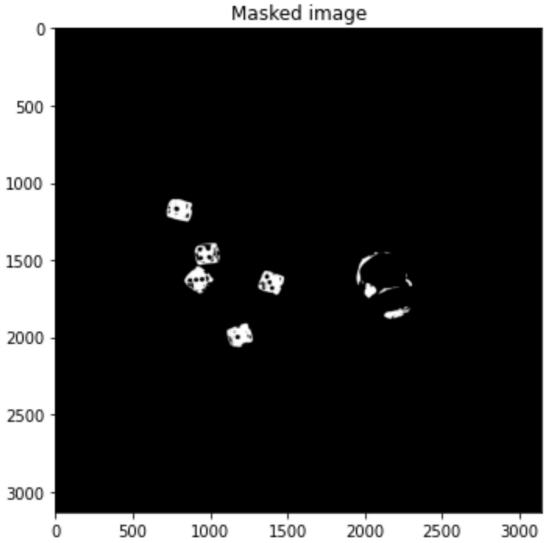


Dice segmentation

Our Approach:

- Since the contrast between the dice and the background colors is high, we can use a hue mask to select the dice
- Afterwards, the connected component labelling algorithm can be used on the mask
- We have to filter the obtained labels to obtain good results

The segmentation from background



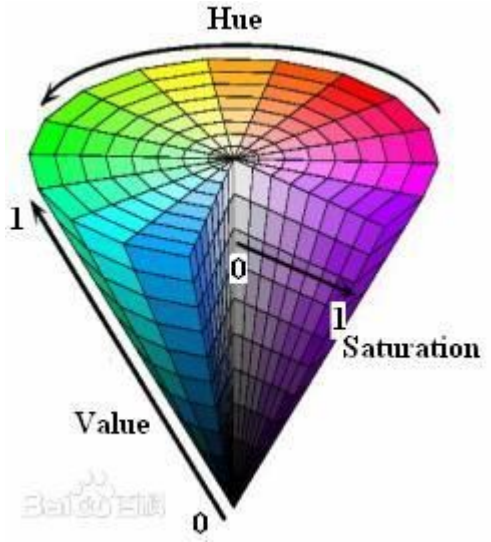
The segmentation from background

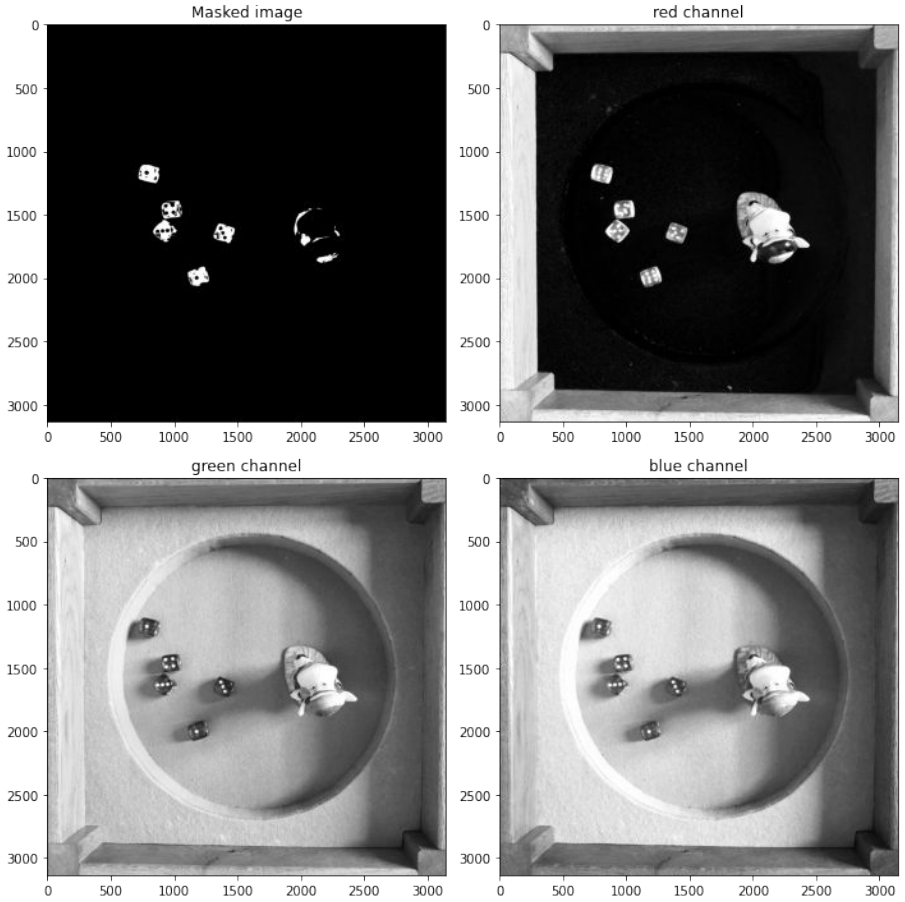
```
def dice_mask(img):
 imhsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
 low_red = np.array([122,50,50])
 high_red = np.array([136,255,255])
 mask = cv2.inRange(imhsv, low_red, high_red)
 return mask
```

We found that using the HSV color space is the best for segmentation. First, we used selection based on the shade of Red on the dices. So, we selected two HSV values to define the lower and upper range of the Reds in the dices and selected those pixels of the image that are in that range.

HSV space and mask

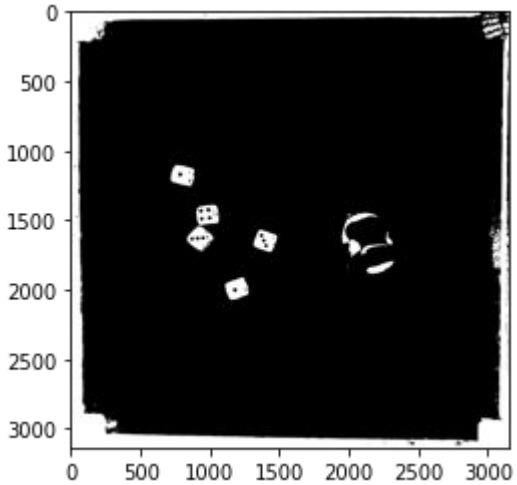
```
The classic steps:
low_red = np.array([122,50,50])
high_red = np.array([136,255,255])
mask1 = cv2.inRange(imhsv, low_red, high_red)


low_red2=np.array([122,100,100])
high_red2=np.array([135,255,255])
mask2=cv2.inRange(imhsv,low_red2,high_red2)

mask=mask1+mask2
```




Masked image
red channel
green channel
blue channel
0
500
1000
1500
2000
2500
3000

Alternative Solution For masking

We experimented with the Opponent Color Space using the conversion formulas that we used during the practical work. With some tweaking we managed to get a more accurate mask.



Dice segmentation

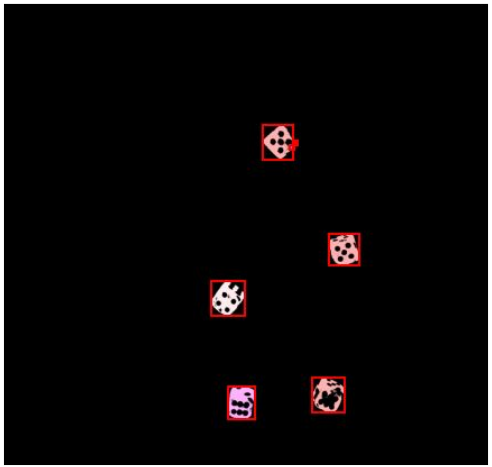
Filtering Labels:

- The bounding box should be large enough, roughly square, and mostly filled in

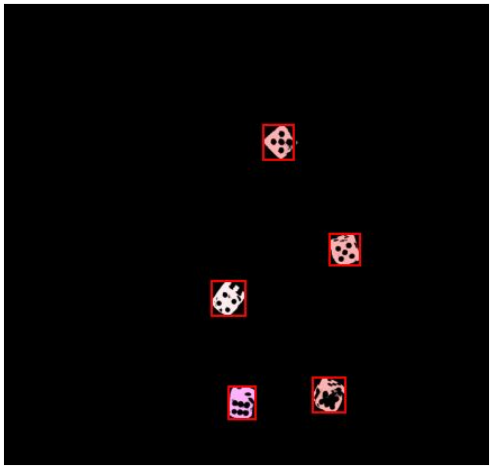
Dice segmentation

Dice segmentation of dices7.jpg

Original



Segmentation, no filter

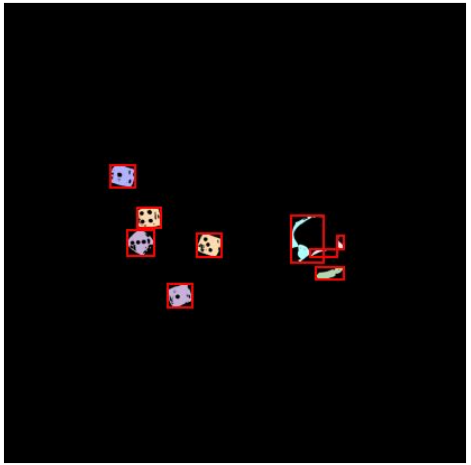


Segmentation, with filter

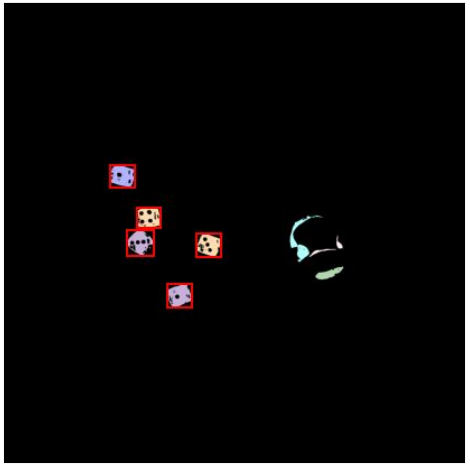
Dice segmentation

Dice segmentation of dices2.jpg

Original



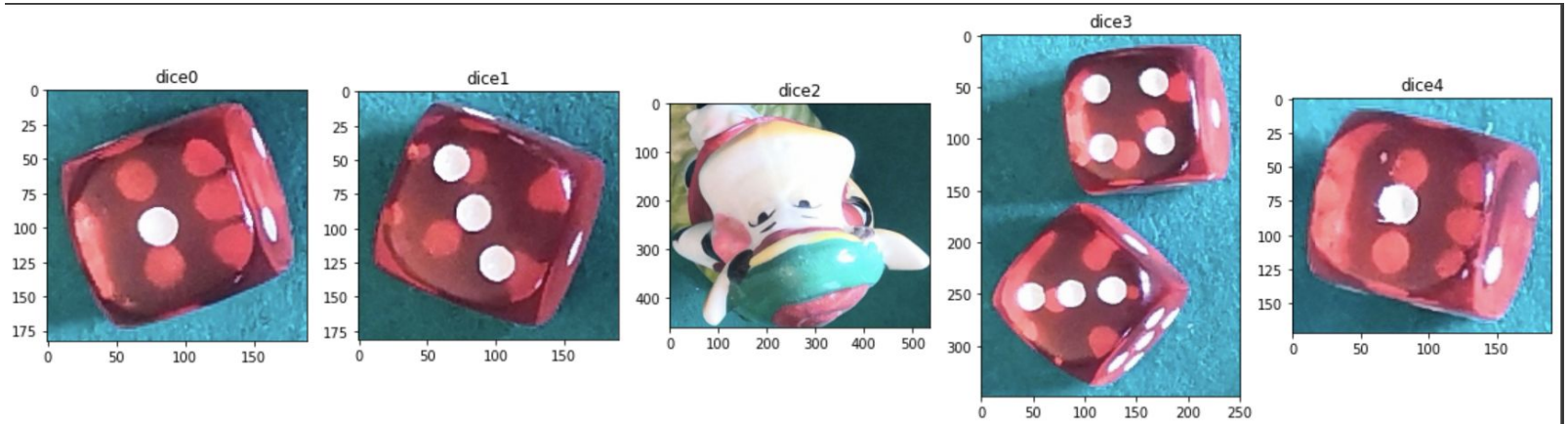
Segmentation, no filter



Segmentation, with filter

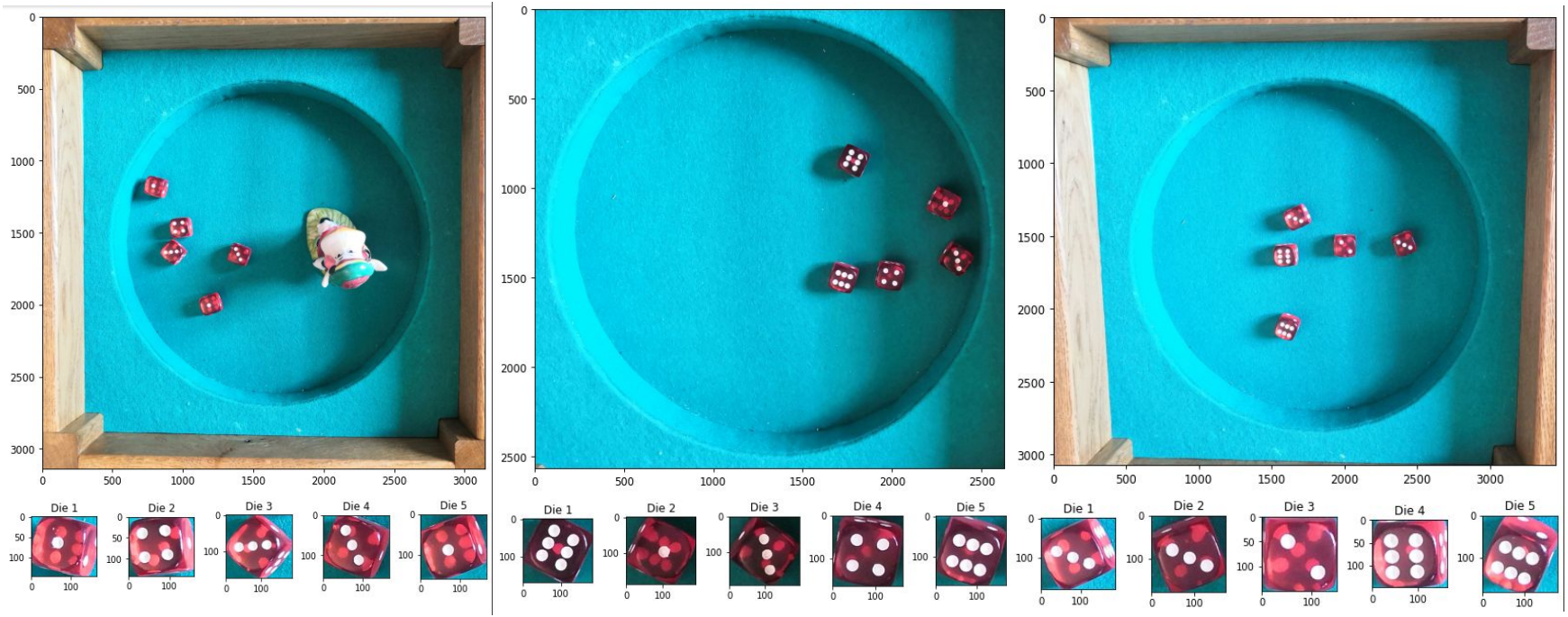
Alternative Methods

- Load image, grayscale, Gaussian blur, Otsu's threshold
- Two pass dilate with horizontal and vertical kernel
- Find contours, filter using contour threshold area, and draw rectangle (cv2.findContours)
- Store the extracted pictures in array to process it later



Dice segmentation

The Algorithm properly separates the dice from the image for all images in the dataset!


Die 1
Die 2
Die 3
Die 4
Die 5
Die 1
Die 2
Die 3
Die 4
Die 5
Die 1
Die 2
Die 3
Die 4
Die 5

Dots detection

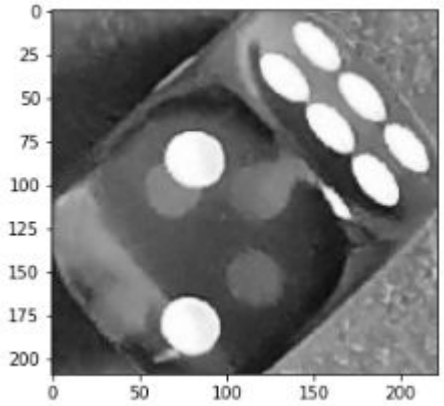
- Hough transform
  - converting to grayscale
  - Gaussian denoising
  - using function from CV2 library: cv2.HoughCircles()
  - extracting coordinates of circles
- combining Hough function with binary mask
- connected component labelling algorithm provided by skimage
- other experiments

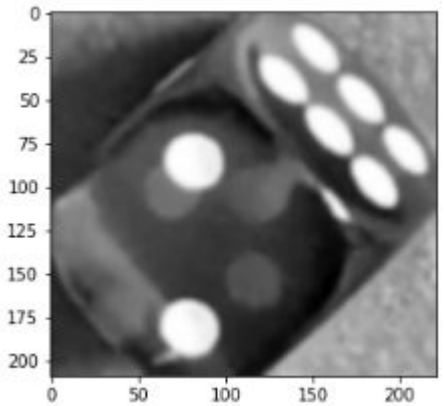
cv2.HoughCircles() - parameters

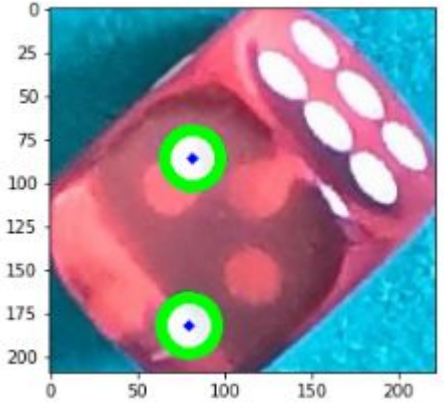
- **method:** HOUGH GRADIENT
- **dp:** inverse ratio of the accumulator resolution to the image resolution (in many examples =1)
- **minDist:** minimum distance between the center *(x, y)* coordinates of detected circles
- **param1:** gradient value used to handle edge detection
- **param2:** accumulator threshold value for the cv2.HOUGH_GRADIENT method
- **minRadius:** minimum size of the radius (in pixels)
- **maxRadius:** maximum size of the radius (in pixels)

MinDist and maxRadius depends on image size. Param1 and param2 were specified empirically.

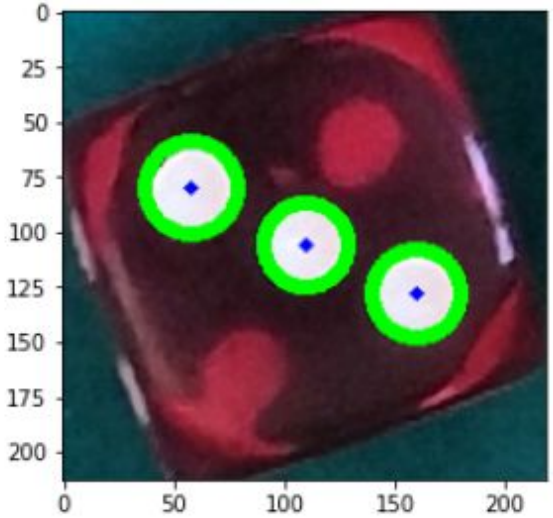
cv2.HoughCircles()



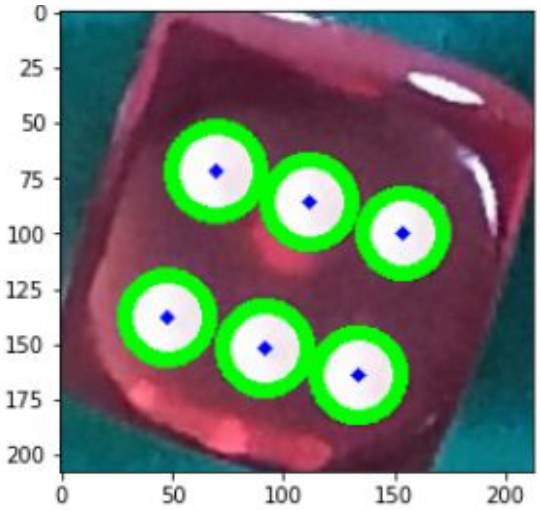




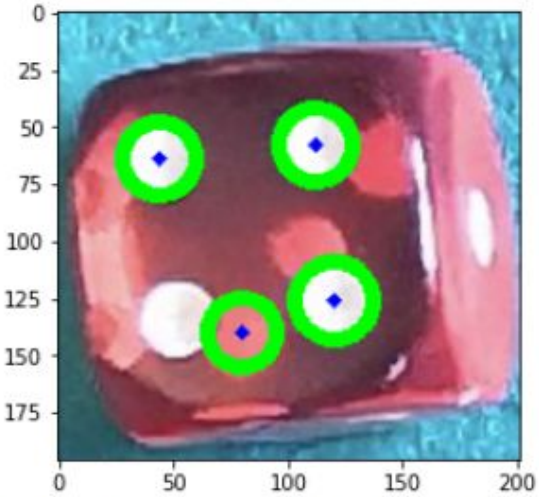
cv2.HoughCircles()



Number of dots on this dice: 3

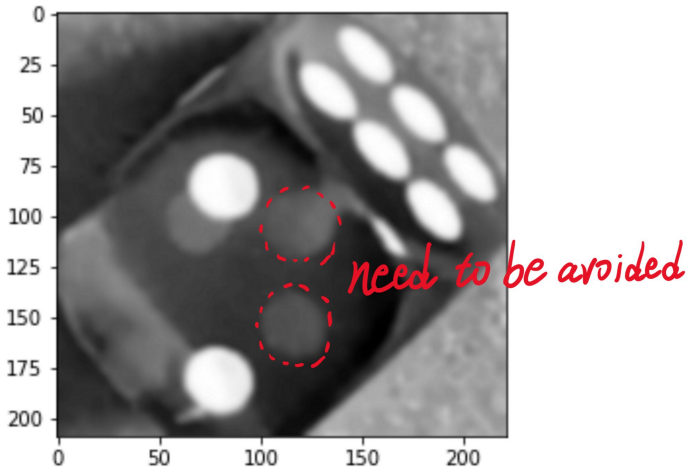


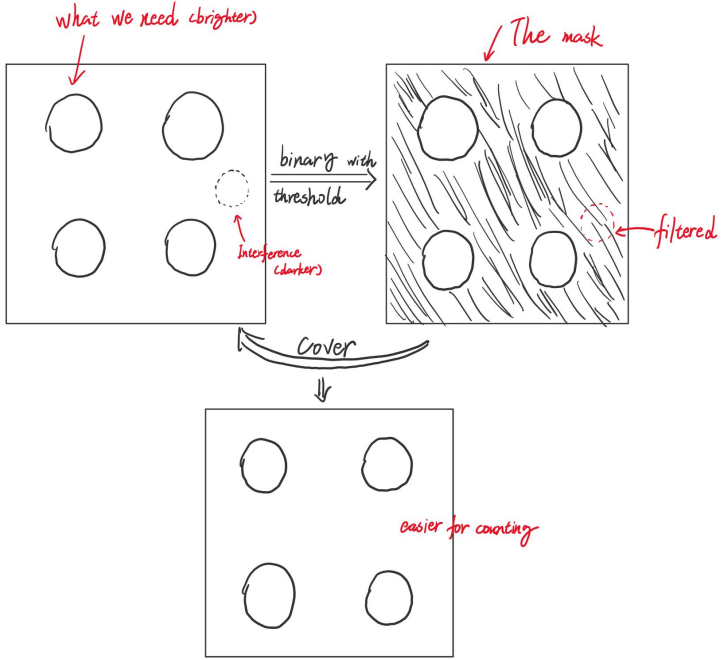
Number of dots on this dice: 6



Number of dots on this dice: 4

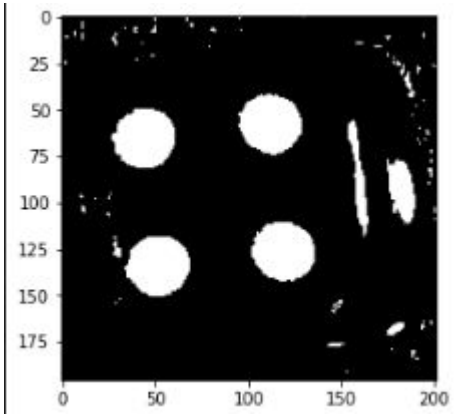
cv2.HoughCircles() + binary mask

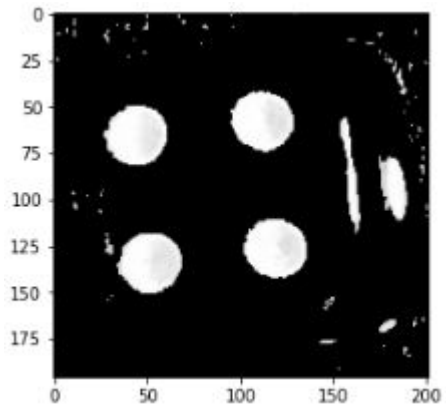


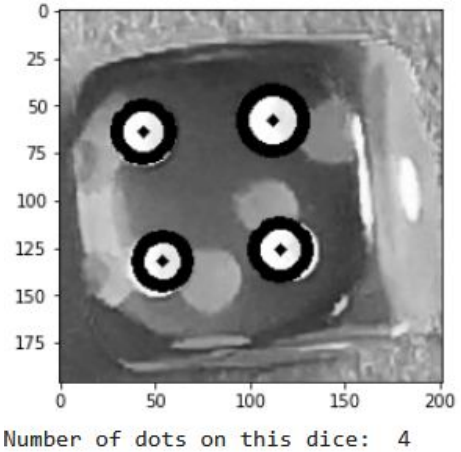


cv2.HoughCircles() + binary mask

- inverse binary mask (threshold= 0.8 * max value from image)
- masked image = image* binary_mask
- applying HoughCircles() on masked image

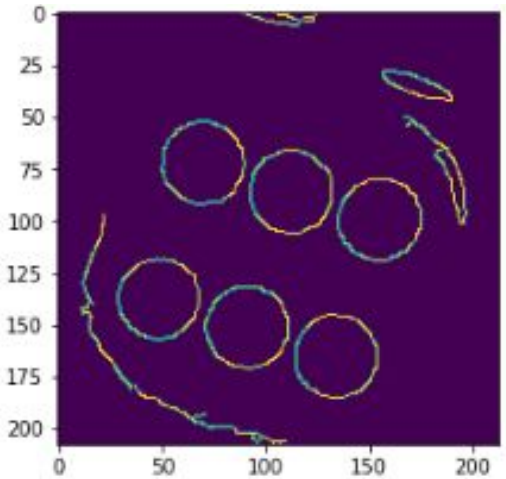






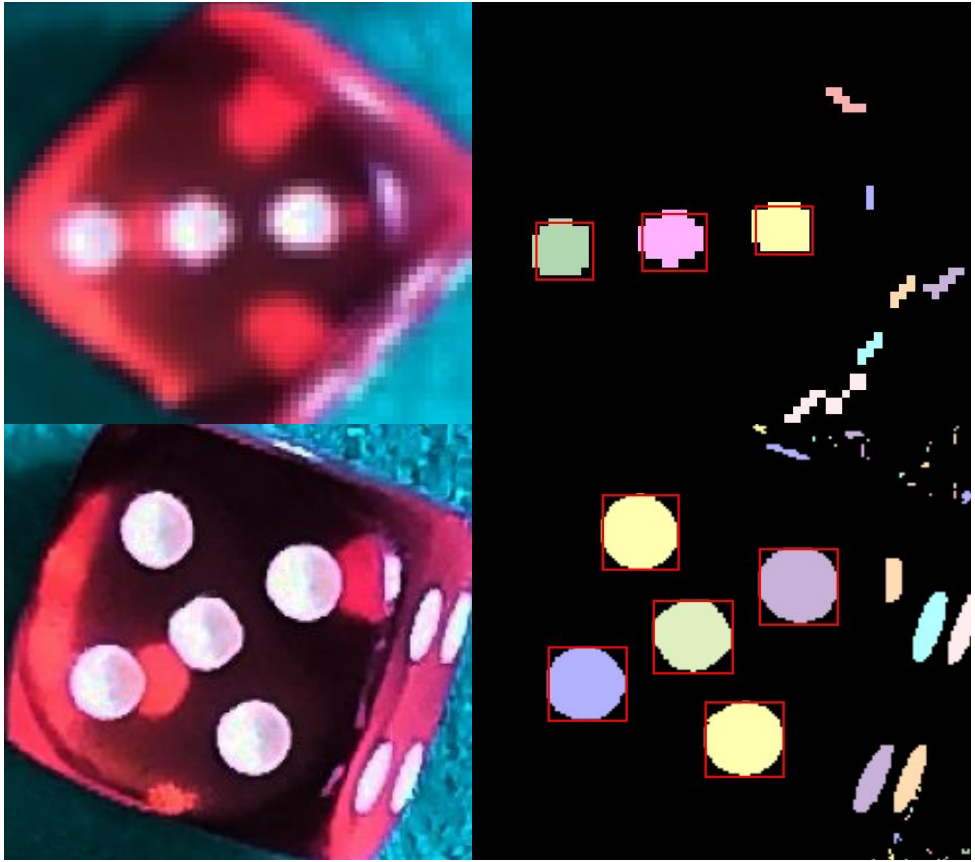
Other Experiments

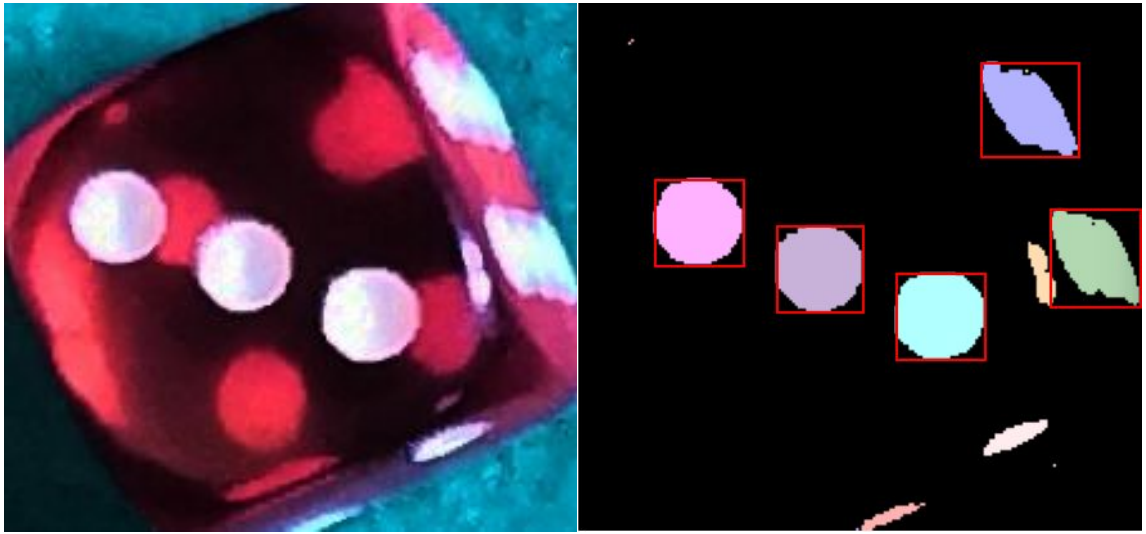
- Canny edge detector
- cv2.findCountours()



Dot Detection

- In the final version, we used another approach based on colorimetry
- On each die, we use saturation reduction and gamma transformation
- We then grayscale the image, use a thresholding and use the labeling method
- Once again, we have to select which labels we will count





Results

Success Rate: 82.5%

- Mistakes occur in low lighting conditions, or when multiple sides of the dice are visible
- In order to improve the accuracy, we have to select only the upper side of the dice and improve the mask for low light images.
- We also could use the fact that the dataset we have contains only 5 dices per image.

What can be improved in the future?

1.The pre-processing before Segmentation

2.The generalization from specialization

3.The probable application