UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

Campus Luiz Meneghel

KAIO FELIPE BARBOSA GARCIA

PROTOCOLO DE PROCESSAMENTO DE IMAGEM - RESULTADO DOS DADOS

Bandeirantes
2022

KAIO FELIPE BARBOSA GARCIA

PROTOCOLO DE PROCESSAMENTO DE IMAGEM - RESULTADO DOS DADOS

Trabalho de graduação do curso de Ciência da Computação, na disciplina de Computação Gráfica, que consiste na criação de um protocolo de reconhecimento de imagem capaz de reconhecer objetos para algum fim, um exemplo seria contá-los. Trabalho resultará em nota para disciplina de Computação Gráfica na Universidade Estadual do Norte do Paraná - *Campus* Bandeirantes.

Professor(a): Prof. Wellington Della Mura.

Bandeirantes

Sumário

0. INTRODUÇÃO
1. IDEIA
2. AQUISIÇÃO
3. PRÉ-PROCESSAMENTO
- 3.1. Resize
- 3.2. Grayscale
- 3.3. Blur

4. SEGMENTAÇÃO
- 4.1. Thresholding
- 4.2. Erosion & Dilatation

5. INTERPRETAÇÃO
- 5.1. Counting Objects
- 5.2. Draw

6. CÓDIGO
- 6.1. Repositório
- 6.2. Bibliotecas
- 6.3. Passo a passo

7. RESULTADOS

0. INTRODUÇÃO

Este trabalho consiste em uma aplicação prática dos conceitos aprendidos nas aulas da disciplina de Computação Gráfica, sobre o tema de Processamento de Imagem. O objetivo é que seja criado um protocolo para o processamento de imagem capaz de reconhecer objetos para algum fim, esse protocolo deve conter os passos que foram realizados para obter sucesso no reconhecimento, dentre eles temos: aquisição, pré-processamento, segmentação, interpretação e resultados.

1. IDEIA

A ideia é que o algoritmo seja capaz mostrar o resultado de uma rolagem de dados de seis lados, por meio de uma foto, ele conseguirá encontrar as bolinhas pretas no topo dos dados e as contá-las para assim dar o resultado, para a situação proposta os dados devem ser brancos com as bolinhas pretas e o fundo deve ser branco (folha de papel).

2. AQUISIÇÃO

A imagem foi obtida via pesquisa no google imagens pois não encontrei dados brancos com bolinhas pretas para fazer a aquisição por câmera, também foi necessário que essa imagem fosse modificada com Photoshop para adicionar mais dados a serem identificados.

Em uma situação real, a foto dos dados deve ser tirada de cima, para que assim somente a face de cima fique visível e é recomendado o uso do *flash* do celular (ou câmera) usado para realizar a captura e uma luz auxiliar (ring-light por exemplo), para que não haja sombras que atrapalhem o reconhecimento. Outra necessidade é um fundo branco (podendo ser uma folha de papel) para que não tenha interferência na leitura e os dados devem ser brancos com as bolinhas pretas. Em caso de outras cores deve-se fazer mudanças no código para as reconhecer. Todas imagens utilizadas estão nesta pasta do repositório: https://github.com/kaiofbgarcia/OpenCV/tree/master/Dados/imgDados**.**

3. PRÉ-PROCESSAMENTO

3.1. Resize

Em casos onde a imagem obtida é grande é necessário redimensionar a imagem para que fique melhor de exibir e reconhecer. Deve-se sempre manter o padrão da imagem, por exemplo: a imagem usada para testes (dadosBrancos3.png) possui dimensões de 2000x1333 (largura x altura, em *pixels*) e para que ela fosse melhor exibida e reconhecida foi utilizado o método *resize* do *opencv* para que ficasse em 666x444.

3.2. Grayscale

Foi aplicado uma transformação da imagem em cinza para que ela pudesse ir para a binarização, depois para tornar a imagem em tons de cinza foi utilizado a função cvtColor do OpenCV e a cor foi COLOR_BGR2GRAY.

3.3. Blur

O blur foi utilizado para deixar a área reconhecida mais “bonita”, segue abaixo imagens de como seria com e sem o uso dessa função:

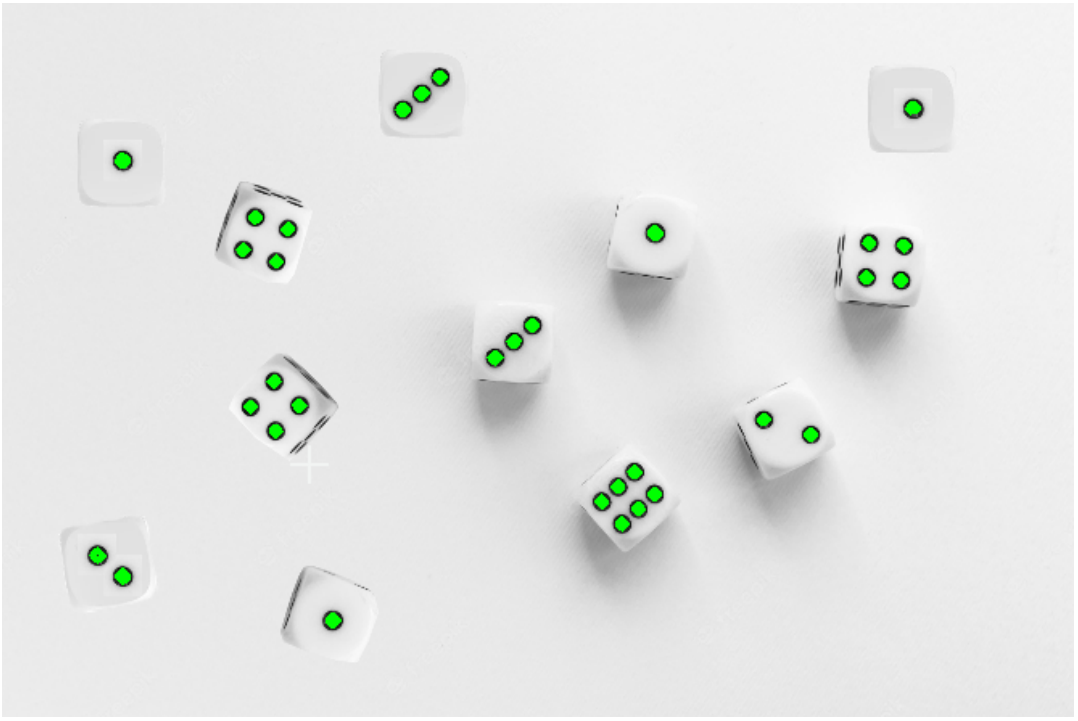


Figura 1: Com o uso do *blur*

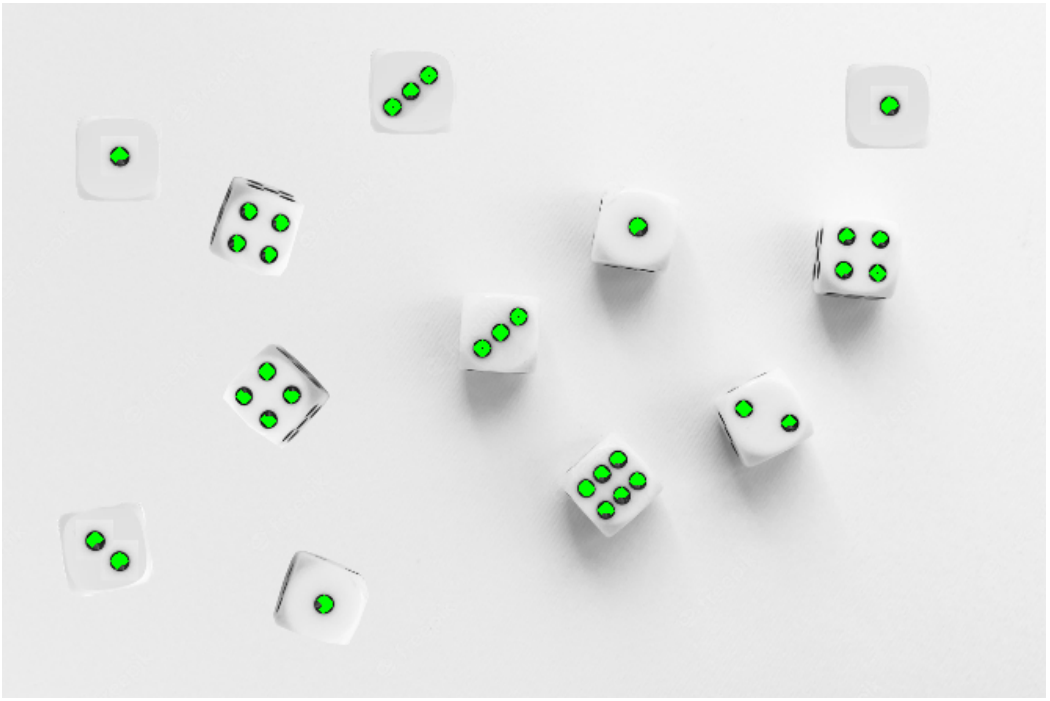


Figura 2: sem o uso do *blur*

A área verde representa o que foi reconhecido pelo algoritmo, como é possível notar sem o uso do blur algumas bolinhas ficam incompletas.

4. SEGMENTAÇÃO

4.1. Thresholding

A binarização (Thresholding) é um método de segmentação simples onde podemos separar uma imagem em regiões de interesse e não interesse a partir de um ponto de corte e essas regiões são representadas pelas cores pretas e brancas. O OpenCV possui uma função para isso, os tipos de binarização usadas foram “ cv2.THRESH_BINARY ”, que deixou a imagem como na Figura 3, e a “ cv2.THRESH_BINARY_INV ”, que serviu para inverter as cores da binarização anterior, Figura 4. Outro exemplo é a binarização de Otsu, mas esta não foi utilizada.

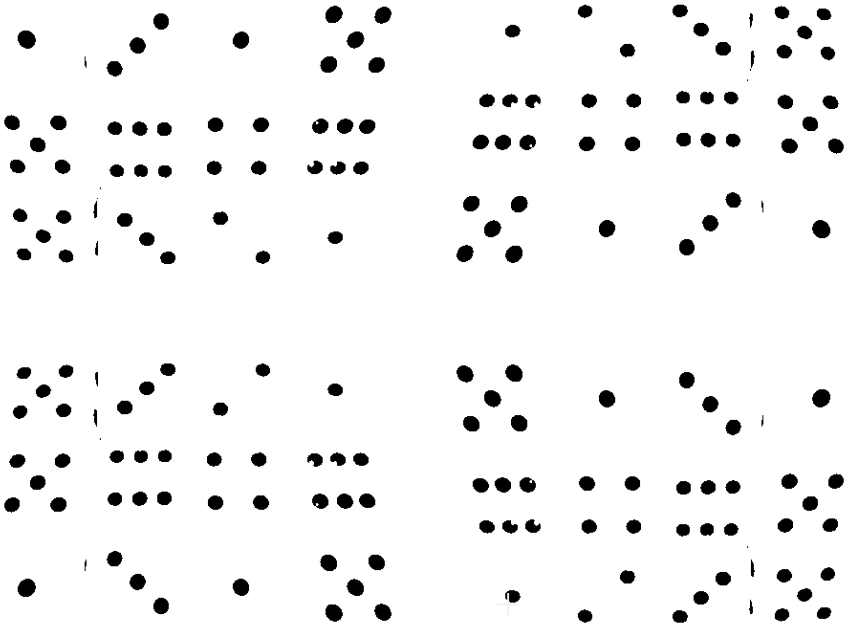


Figura 3: Primeira Binarização

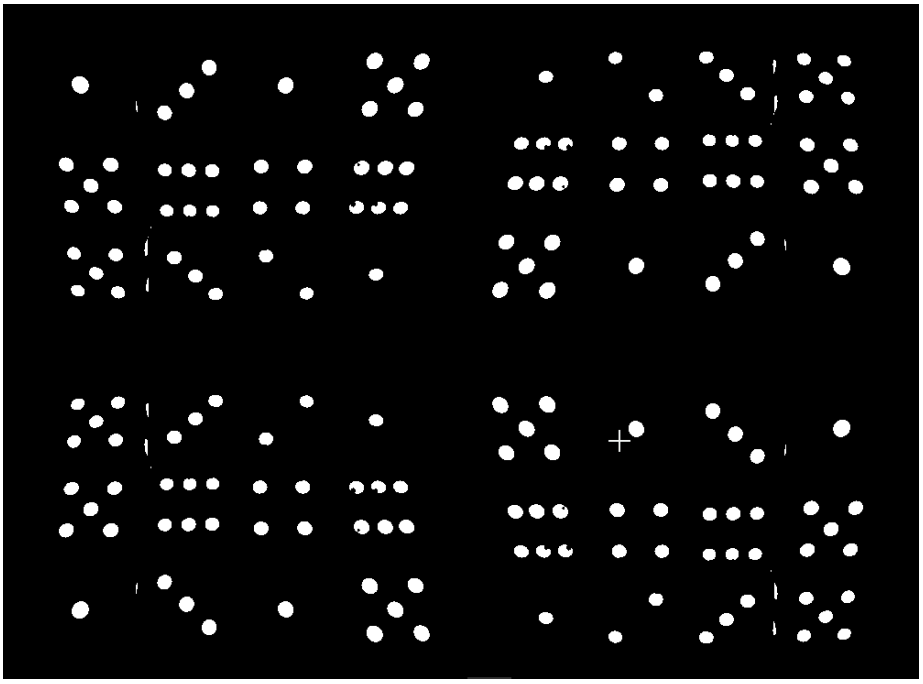


Figura 4: Inversão da Binarização

4.2. Erosion & Dilatation

A função *erode* do OpenCV foi usada para eliminar alguns ruídos que eram reconhecidos como se fossem bolinhas, para eliminá-los sem perder a qualidade dos objetos encontrados foi utilizado o método *dilate*. Estas funções são opostas mas se usadas da maneira certa são muito úteis, a seguir temos imagens de como seria sem usar essas funções.

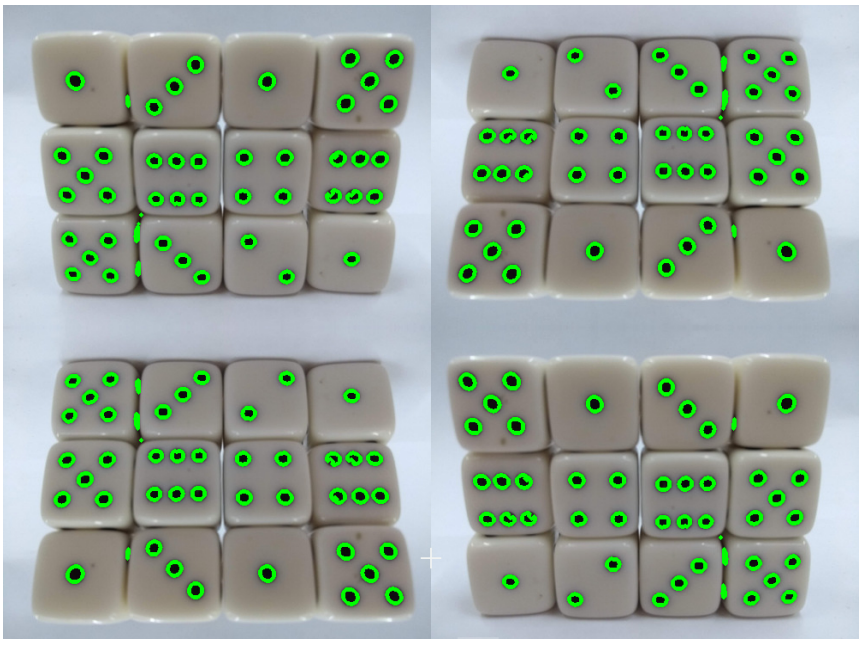


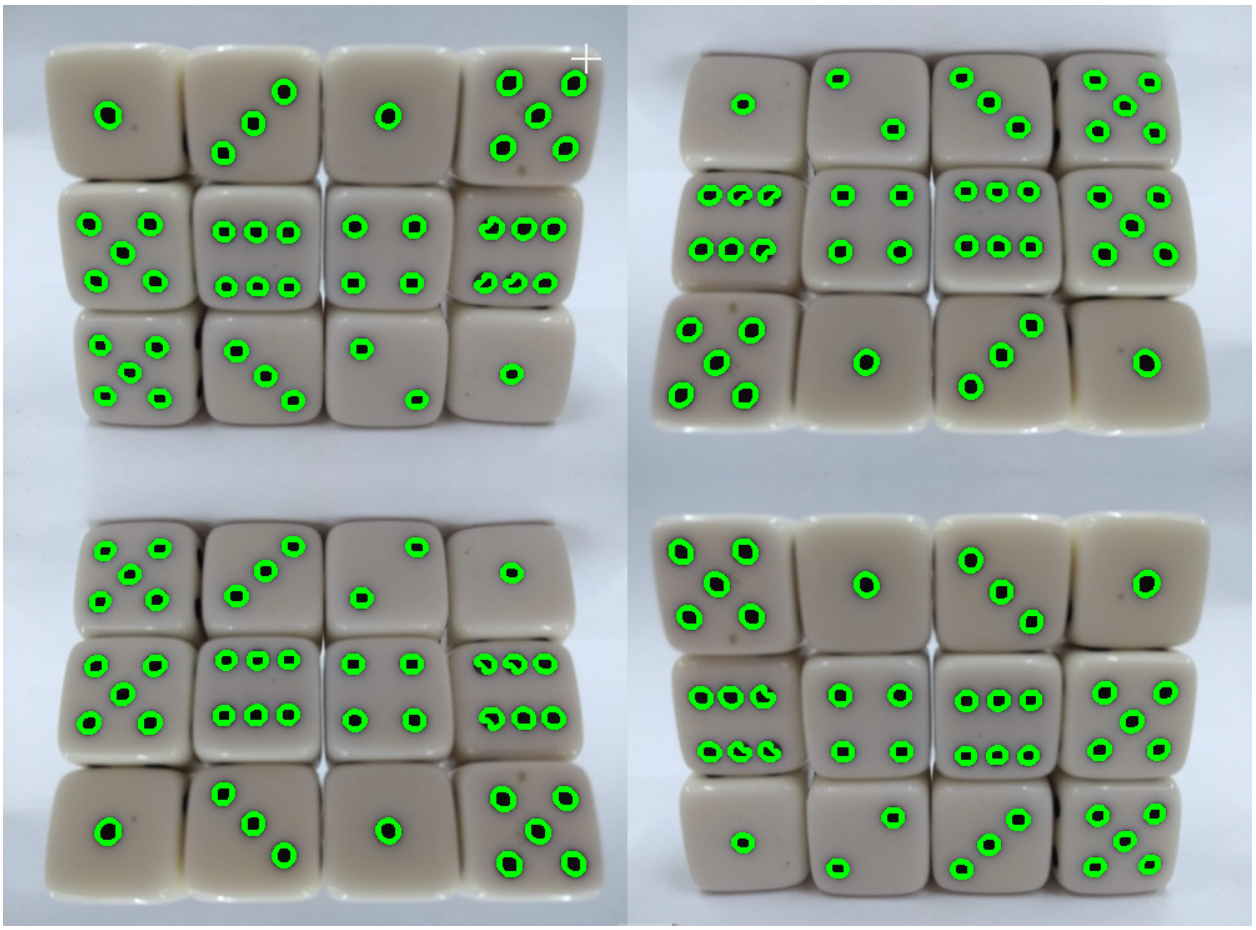
Figura 5: Sem *Erode* & *Dilate*

Como é possível notar alguns dos vãos foram reconhecidos como bolinhas e isso irá atrapalhar na contagem final.

5. INTERPRETAÇÃO

5.1. Counting Objects & Draw

Para realizar as contagens dos objetos foi utilizado a função do OpenCV chamada *findContours*, que faz isso e em seguida vem o *drawContours* que desenha os objetos na imagem original para ser mostrada na tela.



6. CÓDIGO

6.1. Repositório

Todos os códigos, imagens e recursos usados para criar o protocolo estão publicados no seguinte repositório do GitHub: https://github.com/kaiofbgarcia/OpenCV/tree/master/Dados

6.2. Bibliotecas

Os códigos que estão presentes no repositório foram escritos na linguagem Python e as bibliotecas que foram utilizadas foram o OpenCV e o Numpy. Há dois arquivos principais, um utiliza uma imagem onde os dados presentes foram desenhados (*dadosBrancos3.py*) e uma onde

a imagem utilizada aparenta ser de dados reais (*dadosPilha2.py*).

6.3. Passo a passo

A seguir temos um passo a passo de como foi escrito o código *dadosPilha2.py*:

Passo 1: Importação das Bibliotecas

```
import cv2
from cv2 import THRESH_BINARY
from cv2 import erode
import numpy as np
from numpy import binary_repr
```

Como citado anteriormente, estas foram as bibliotecas utilizadas.

Passo 2: Carregar a imagem e fazer o pré-processamento

```
imagem = cv2.imread("imgDados/dadosPilha.jpg")     # Carregar a imagem
gray = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)    # GrayScale
blur = cv2.blur(gray,(3,3))                        # Blur
```

Passo 3: Segmentação

```
ret,binary = cv2.threshold(blur , 30 ,255, cv2.THRESH_BINARY)        # Binarização
ret1,binary1 = cv2.threshold(binary,127,255,cv2.THRESH_BINARY_INV)   # Inverter a Binarização

kernel = np.ones((5,5), np.uint8)                   # Erosão
erosion = cv2.erode(binary1, kernel, iterations=1)  # Erosão
dilation = cv2.dilate(erosion, kernel, iterations=1) # Dilatação
```

Na primeira binarização o ponto de corte é aquele valor 30, que foi variado dependendo da imagem em questão. Por exemplo no *dadosBrancos3.py* o ponto de corte foi definido como 80 pois seu fundo é mais branco.

Passo 4: Interpretação

```
contours, hierarchy = cv2.findContours(dilation, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE) # Procura os Objetos
cont = len(contours)                                                                      # Conta os Objetos

cv2.drawContours(imagem, contours, -1, (0,255,0), 3)     # Desenha os Objetos
```

Aqui é onde foi feita a contagem dos objetos e foi passado para realizar o desenho na imagem original.

Passo 5: Exibição dos Resultados e Imagens

```
print("Resultado dos dados: ", cont)  # Mostra o resultado

cv2.imshow("Dilatation", imagem)      # Mostra a imagem(com o desenho já)

cv2.waitKey(0)                        # Encerra o processo
```

Aqui tem uma parte que seria mais para realizar e visualizar melhor os testes e substituiria o código da imagem anterior:

```
cv2.imshow("Grayscale", gray)            # Mostra a imagem cinza
cv2.imshow("Blur", blur)                 # Mostra a imagem borrada
cv2.imshow("Threshold", binary)          # Mostra a imagem primeira binarização
cv2.imshow("Threshold Inv", binary1)     # Mostra a imagem segunda binarização
cv2.imshow("Erode", erosion)             # Mostra a imagem apos erosão
cv2.imshow("Dilatation", dilation)       # Mostra a imagem apos dilatação
cv2.imshow("Draw", imagem)               # Mostra a imagem(com o desenho já)

cv2.waitKey(0)
```

7. RESULTADOS

Os algoritmos criados obtiveram os resultados esperados, contando com sucesso o valor dos dados das imagens escolhidas, a seguir temos algumas imagens que foram analisadas com sucesso, nelas os desenhos destacados em verde representam os objetos encontrados e uma imagem do terminal que apresenta o resultado dos objetos encontrados, ou seja, o valor resultante dos dados.

Figura 6: Imagem Resultado do *dadosBrancos3.py*

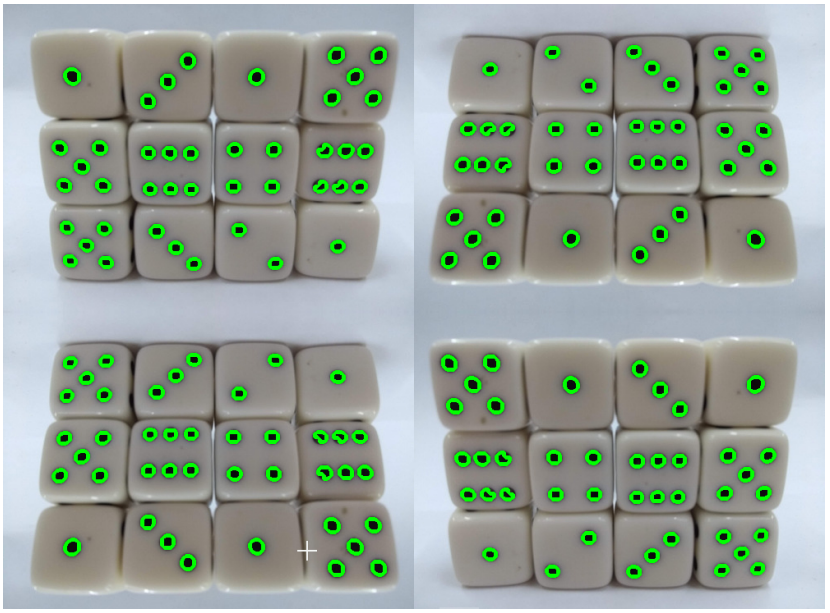


Figura 7: Imagem Resultado do *dadosPilha2.py*

```
PS C:\Users\kaiof\Desktop\Computação Gráfica\OpenCV\Dados> python .\dadosBrancos3.py
Resultado dos dados:  32
PS C:\Users\kaiof\Desktop\Computação Gráfica\OpenCV\Dados> python .\dadosPilha2.py
Resultado dos dados:  168
PS C:\Users\kaiof\Desktop\Computação Gráfica\OpenCV\Dados>
```

Figura 8: Terminal com os Resultados dos Dados