Trabalho 1

MC920 Introdução ao Processamento Digital de Imagens

Rafael Figueiredo Prudencio
r186145@g.unicamp.br

28 de Março de 2018

1 Introdução

Este trabalho visa realizar alguns processamentos básicos em imagens experimentando com novas bibliotecas do Python. O trabalho consiste na realização de três tarefas: a obtenção do contorno dos objetos de uma imagem, o cálculo da área e do perímetro dos objetos de uma imagem com a respectiva rotulação na imagem original e a obtenção de um histograma da área dos objetos detectados.

2 Tarefas

2.1 Contorno dos Objetos

Para obter o contorno dos objetos, basta chamar a função *edges* implementada no código anexo. A partir da imagem original, a função retorna a imagem com apenas os contornos de cada objeto destacados em vermelho. Usando o pacote *cv2* do OpenCV foi possível completar a tarefa pedida com uma única linha de código:

edges = cv2.Canny(image, threshold1=9000, threshold2=12000, apertureSize = 5)

A função *cv2.Canny* admite 5 argumentos, sendo 2 opcionais: *image* é uma matriz representando a imagem onde deseja-se aplicar o algoritmo, *threshold1* é o limite inferior dos pixels que serão considerados bordas, *threshold2* é o limite superior desses pixels, o argumento opcional apertureSize indica a dimensão do kernel usado no operador Sobel e *L2gradient* é um booleano indicando qual equação deve-se usar para calcular o gradiente em cada pixel. Nas equações 1 e 2, G_x é o resultado da convolução do Sobel kernel para mudanças horizontais com a imagem original e G_y corresponde ao resultado com o Sobel kernel para mudanças verticais. Os limites superior e inferior são usados para filtrar quais bordas devem ser consideradas. Se houver um ponto de uma borda com gradiente maior ou igual ao limite superior e os demais pontos dessa borda com o gradiente maior que o limite inferior, essa borda será considerada válida e os pixels correspondentes a ela passam a valer 0. Caso contrário, a borda é descartada e seus pixels valem 255. Por default, a biblioteca usa um kernel de dimensão 3 para o operador Sobel e o gradiente L1 que é menos custoso de implementar. A escolha dos parâmetros foi feita empiricamente. Buscou-se eliminar a redundância na detecção das bordas e por isso usou-se um valor alto para os dois limites. Após a obtenção da imagem binária retornada pela função *cv2.Canny*, substituiu-se os pixels pretos da imagem por pixels vermelhos transformando-a para o modelo RGB. O resultado da função para a imagem *objetos2.png* é dado pela Figura 1.

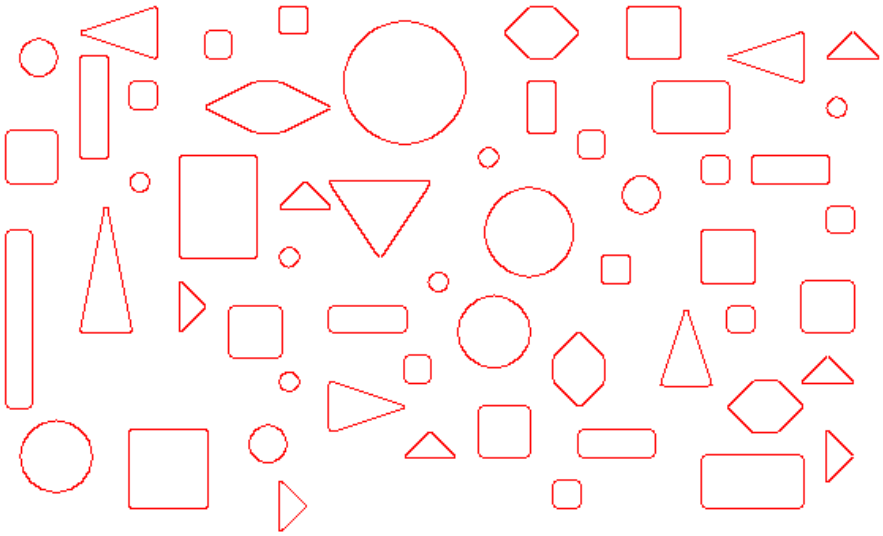


Figura 1: Contorno dos objetos detectados na imagem obtido através do Canny edge detector. A imagem acima foi gerada pela função *edges* do módulo *object.py* com a opção *-s* de execução.

A função implementada é bem limitada com relação ao tipo de imagem onde obtém-se um resultado satisfatório. Os contornos detectados têm que possuir um gradiente muito elevado, ou seja, a discrepância de cor deve ser alta para que o algoritmo identifique a borda do objeto já que os limites superior e inferior usados na função *cv2.Canny* são bem elevados.

$$G = \sqrt{G_x^2 + G_y^2} \tag{1}$$

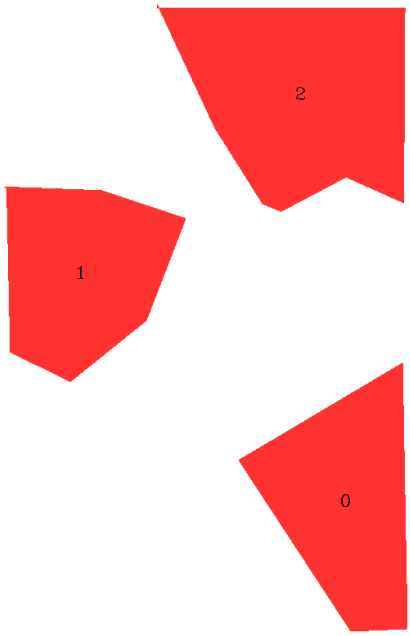
$$G = |G_x| + |G_y| \tag{2}$$

2.2 Extração de Propriedades dos Objetos

A extração de propriedades foi implementada na função *obj_properties* do código anexo. A partir da imagem original, a função imprime na saída padrão a lista dos objetos seu índice, área e perímetro e salva ou exibe uma imagem com cada objeto rotulado de acordo com o índice impresso na saída padrão. A função retorna uma a imagem rotulada e a lista de áreas dos objetos.

No código, primeiro verifica-se quantos canais a imagem possui. No caso de uma imagem de três canais, assume-se que é uma imagem RGB e aplica-se a conversão para uma imagem monocromática usando a função *cv2.cvtColor*. Uma das limitações do código é que se a imagem passada possuir três canais mas usar um modelo de cor diferente como HSV, o programa seguirá executando sem erros mas com um resultado final indesejável.

Com a imagem monocromática do passo anterior como parâmetro, chama-se a função *cv2.threshold* para converter a imagem com possivelmente diferentes níveis de cinza para uma imagem binária com pixels valendo 0 ou 255. Com a imagem binária como parâmetro, chama-se a função *cv2.findContours* para obter uma lista dos contornos de cada objeto detectado. A partir dos contornos de cada objeto pode-se chamar as funções *cv2.contourArea* e *cv2.arcLength* para



(a) Regiões rotuladas

```
Terminal
rafael:~/Documents/unicamp/mc/920/t1
$ ./object.py -s data/convex.png
número de regiões: 3
região:  0   perímetro: 916    área:41112
região:  1   perímetro: 805    área:38346
região:  2   perímetro:1054    área:56669
rafael:~/Documents/unicamp/mc/920/t1
$
```

(b) Propriedades

Figura 2: Resultado completo da função *obj_properties*. (esquerda) Imagem com regiões rotuladas. (direita) Saída do terminal com a área e o perímetro calculados para cada região.

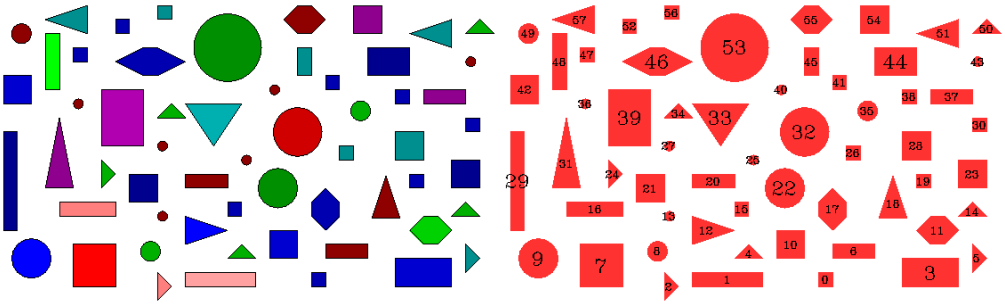
obter a área e o perímetro de cada objeto detectado. Esses valores arredondados para o inteiro mais próximo são impressos na saída padrão, como indicado pela Figura 2b.

Finalmente, rotulou-se todos objetos detectados na imagem. Primeiro, calculou-se o centróide de cada contorno usando-se a função *cv2.moments* e a partir dos momentos calculou-se a posição do centróide (C_x, C_y) usando as equações 3 e 4. Como a função *cv2.putText* precisa da posição do canto inferior esquerdo da string que irá sobrepor a imagem, foi necessário calcular o centro do texto que pretendia-se escrever usando a função *cv2.getTextSize* que retorna as dimensões da caixa de texto. Com o centróide do objeto e as dimensões da caixa de texto é possível indicar à função *cv2.putText* o ponto onde deve começar a escrita do texto para que ele fique centralizado no objeto. Como a origem da imagem está no canto superior esquerdo e o eixo y cresce para baixo e o x para a direita, basta subtrair metade da largura da caixa de texto de C_x e somar metade da altura da caixa de texto a C_y. O resultado da rotulação dos objetos da imagem na Figura 3a é dado pela Figura 3b.

$$C_x = \frac{M_{10}}{M_{00}} \tag{3}$$

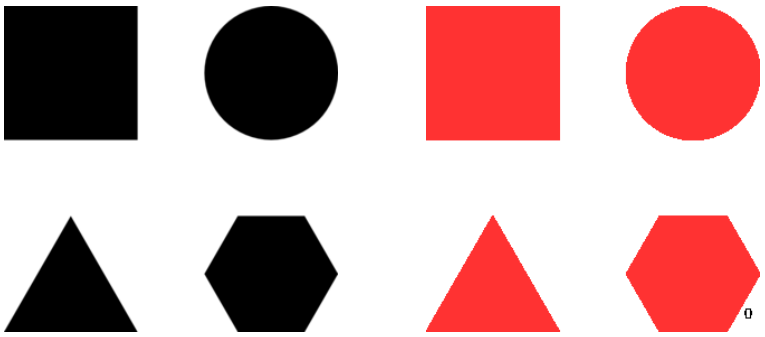
$$C_y = \frac{M_{01}}{M_{00}} \tag{4}$$

Uma das limitações do código é que ele falha na tentativa de rotular objetos que têm seu contorno na borda da imagem. Como um dos contornos retornados pela função *cv2.Contours* é o



(a) Original *(b)* Regiões rotuladas

Figura 3: Resultado parcial da função *obj_properteis* no código anexo. (esquerda) Imagem original obtida pelo site da disciplina. (direita) Imagem com cada região detectado com seu respectivo rótulo.



(a) Original *(b)* Regiões rotuladas

Figura 4: Resultado parcial da função *obj_properteis* no código anexo. (esquerda) Imagem original de formas geométricas cujos extremos estão na borda da imagem. (direita) Imagem com apenas um rótulo detectado entre um dos objetos e o canto inferior direito da imagem.

contorno da imagem inteira e esse contorno não nos interessa, o método escolhido para descartá-lo foi remover todos os contornos com pixels na borda da imagem da lista de contornos válidos. Dessa forma, se um objeto tem suas extremidades na borda da imagem, ele será descartado e o programa irá falhar, como pode ser visto na rotulação da Figura 4.

2.3 Histograma de Área dos Objetos

O histograma de área dos objetos foi implementado na função *histogram* do código anexo. Essa função tem como parâmetro o resultado do cáculo das áreas da função *edges* da seção 2.2. A partir da lista de áreas, a função primeiro define as separações do histograma verificando qual o objeto de maior área da lista e criando os respectivos "bins"para objetos pequenos (área < 1500), médios (1500 $\leq$ área < 3000) e grandes (área $\geq$ 3000). Usando a função *matplotlib.pyplot.hist* com a lista de áreas e os bins como parâmetros, plotou-se o histograma dado pela Figura 5.

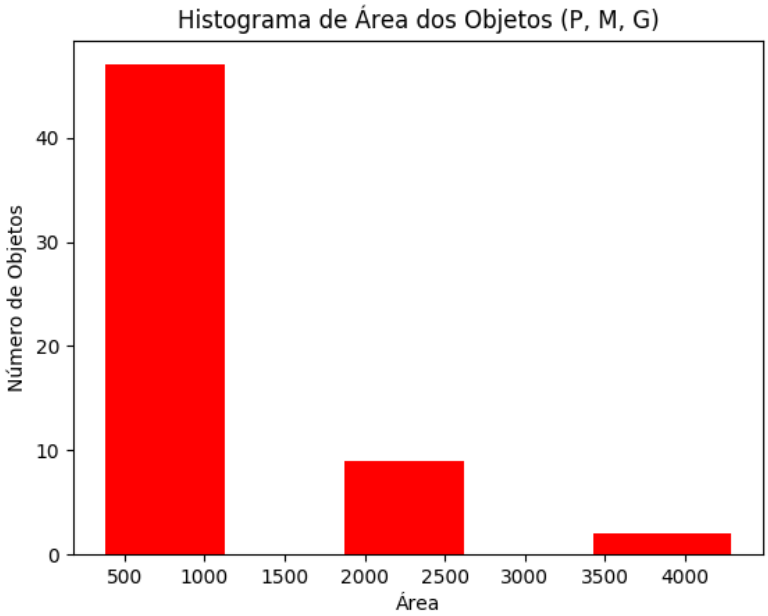


Figura 5: Histograma de área dos objetos detectados na imagem da Figura 3. As colunas da esquerda para direita correspondem, respectivamente, aos objetos classificados como pequenos, médios e grandes.

3 Execução

O código, implementado em Python, pode ser rodado como script através do terminal com um número qualquer de arquivos de imagens como parâmetros. Uma execução simples para processar e exibir duas imagens *fig1.png* e *fig2.png* seria:

./objects.py fig1.png fig2.png

O usuário também pode configurar a execução do programa através de duas flags (*./objects.py - -help* para mais detalhes): *-s* e *-d*. A flag -s sinaliza ao programa que deve salvar as imagens geradas ao invés de exibí-las e a flag -d permite com que o usuário especifique o diretório onde deseja salvar essas imagens (por default elas são salvas no diretório *./out*). Um exemplo de execução customizada seria executar a linha:

./objects.py -s -d sadia fig1.png fig2.png fig3.png

Algumas limitações em relação a execução são que o usuário precisa explicitar o caminho para todas as imagens que deseja processar e não pode, por exemplo, passar um diretório de imagens como parâmetro. Além disso, a código só aceita imagens PNG e não permite que o usuário mude nome dos arquivos que serão salvos quando a flag - *-save* for usada.

O programa foi rodado com os pacotes NumPy(v1.14.1), Matplotlib(v2.1.1), Scipy(v1.0.0) OpenCV(3.4.0.12) usando Python 3.5.2. Não é garantido o funcionamento do programa em qualquer outro ambiente com diferentes versões além das citadas acima.

4 Conclusão

Através deste trabalho, cumpriu-se os objetivos de realizar as tarefas e familiarizar-se com novos pacotes, como o OpenCV, que serão úteis para o resto da disciplina. O OpenCV foi usado para realizar tanto a tarefa de destacar as bordas através da função *cv2.Canny* como a de identificar as propriedades dos objetos através da função *cv2.findContours*. Por ser uma biblioteca popular com suporte para C, C++ e Python, há uma documentação muito boa disponível no site oficial e várias discussões em fóruns. O histograma foi feito usando a função *matplotlib.pyplot.hist* do Matplotlib que, como o OpenCV, possui boa documentação e é muito popular.

Referências

NumPy v1.14 Manual (2018). URL: `https://docs.scipy.org/doc/numpy/` (acedido em 03/11/2018).

OpenCV 3.4.0-dev documentation (2018). URL: `https://docs.opencv.org/3.4.0/genindex.html` (acedido em 03/11/2017).

Python 3.5.2 (2016). URL: `https://www.python.org/downloads/release/python-352/` (acedido em 03/11/2018).

SciPy 1.0.0 Release Notes (2017). URL: `https://docs.scipy.org/doc/scipy/reference/release.1.0.0.html` (acedido em 03/11/2018).

The Pyplot API (2017). URL: `https://matplotlib.org/2.1.1/api/pyplot_summary.html` (acedido em 03/10/2018).