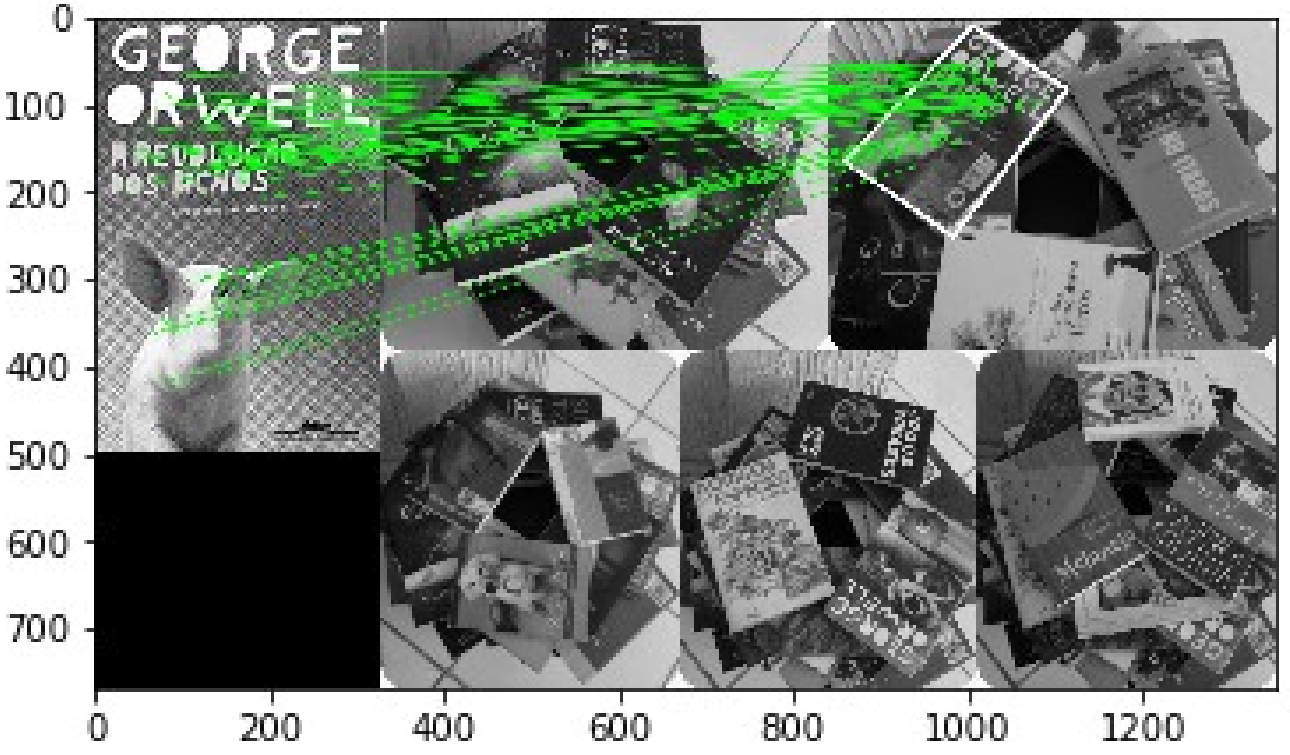
Tarefa de SIFT



Detectando os keyPoints

- Depois da leitura da imagem “livro1.png” e sua conversão em escala de cinza:

```
# inicializa o SIFT e detecta os keypoints
sift1 = cv2.xfeatures2d.SIFT_create()
kp1 = sift.detect(gray1,None)

# os keypoints são pontos salientes, mas não são características. O kp1 basicamente fornece as localizações
```

Desenhando os keyPoints

- A função abaixo desenha bolinhas nas posições nas quais os keypoints foram detectados

```
cv2.drawKeypoints(gray1, kp1, img1)
cv2.imwrite('sift_keypoints1.jpg', img1)
```

Desenhando os keyPoints

- Pode-se usar o parâmetro que fará com que o desenho das bolinhas já indique a orientação e o tamanho do keyPoint

```
cv2.drawKeypoints(gray1, kp1, img1)
cv2.imwrite('sift_keypoints1.jpg', img1)
```

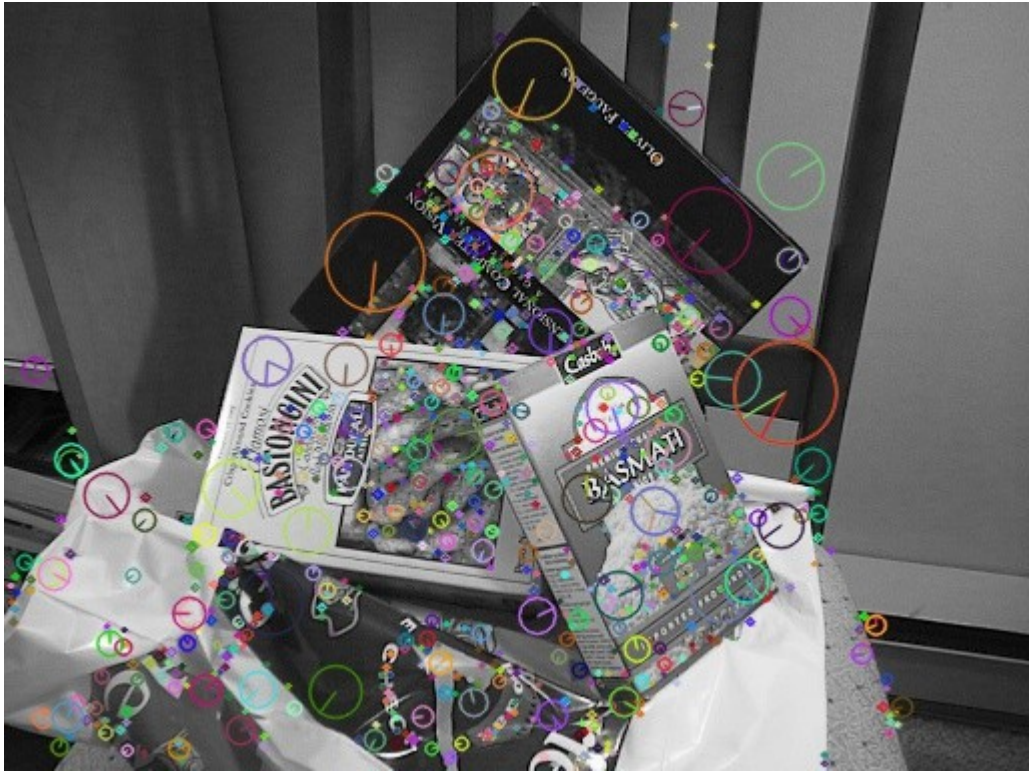
Calculando as Features

- Agora, pode-se usar função que transforma cada keyPoint em uma feature (vetor de características) de tamanho 128

```
kp1, des1 = sift.detectAndCompute(gray1,None)
```

Comparando duas imagens

- Faça o mesmo procedimento que foi feito anteriormente para “livros1.png” (repare no plural)



Feature Matching (Força Bruta)

- Brute-Force matcher (BFMatcher): calcula a distância de uma feature de uma imagem com todas as outras features da outra imagem. Retornam-se os dados dos duas features de menor distância.
- bf = cv2.BFMatcher()
  - Parâmetro 1: default é cv2.NORM_L2, que é bom para SIFT e SURF (cv2.NORM_L1 também é bom). Para vetores binários como os descritores de ORB, BRIEF e BRISK é melhor o cv2.NORM_HAMMING.
  - Parâmetro 2: variável booleana, crossCheck é falso por padrão. Se for verdade, o Matcher retorna apenas as correspondências com valor (i, j), de modo que o i-ésimo descritor no conjunto A tenha j-ésimo descritor no conjunto B como a melhor combinação e vice-versa. Ou seja, as duas características em ambos os conjuntos devem corresponder uns aos outros.

Feature Matching (Força Bruta)

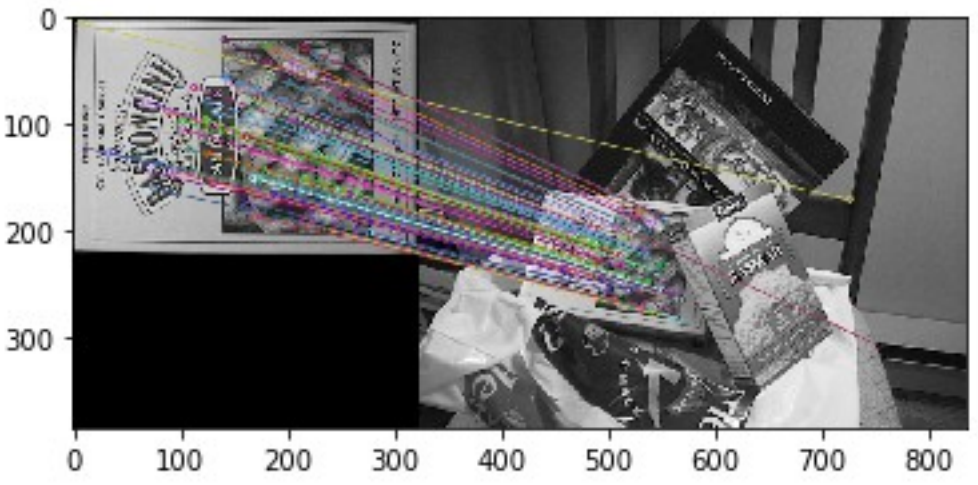
- Depois, pode-se usa a função FMatcher.match () ou a função BFMatcher.knnMatch ().
  - O primeiro retorna a melhor combinação (menor distância).
  - O segundo método retorna k melhores combinações, onde k é especificado pelo usuário.

Desenhando os Matches

- cv2.drawMatches ()
- Desenhar as combinações.
- Coloca lado a lado duas imagens horizontalmente e desenha linhas da primeira imagem para a segunda imagem, mostrando melhores correspondências.
- Há também cv2.drawMatchesKnn que desenha todas as melhores combinações k. Se k = 2, desenhará duas linhas de correspondência para cada ponto chave.

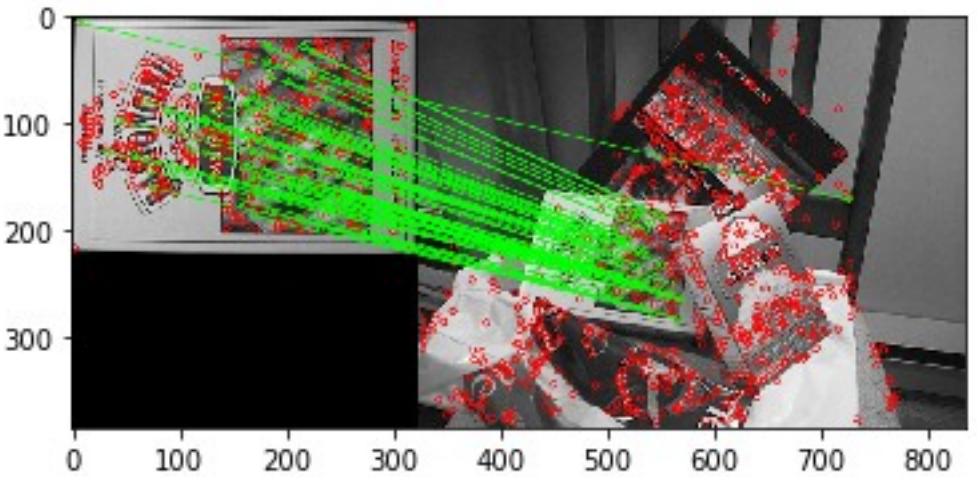
Desenhando os Matches

```
# BFMatcher with default params
bf = cv2.BFMatcher()
#bf = cv2.BFMatcher(cv2.NORM_L1,crossCheck=False)
matches = bf.knnMatch(des1,des2, k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])

# cv2.drawMatchesKnn expects list of lists as matches.
img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None, flags=2)

plt.imshow(img3),plt.show()
```



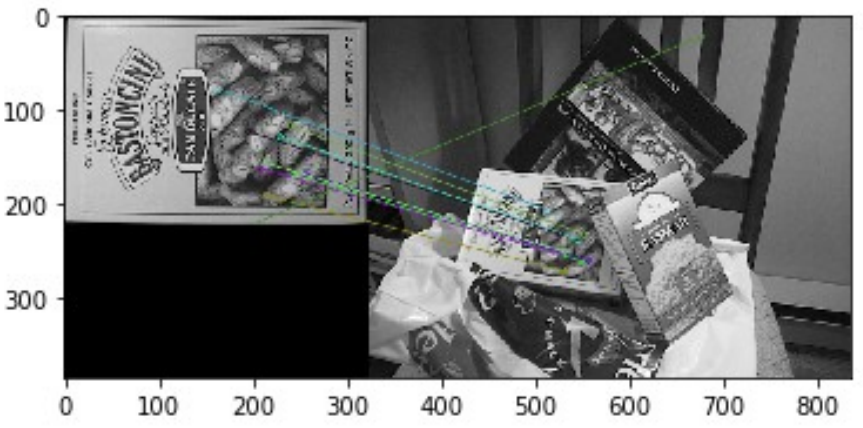
Desenhando os Matches

```
# FLANN parameters
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)

# Need to draw only good matches, so create a mask
matchesMask = [[0,0] for i in range(len(matches))]

# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.7*n.distance:
        matchesMask[i]=[1,0]

draw_params = dict(matchColor = (0,255,0), singlePointColor = (255,0,0), matchesMask = matchesMask, flags = 0)

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,matches,None,**draw_params)
plt.imshow(img3),plt.show()
```



Desenhando os Matches

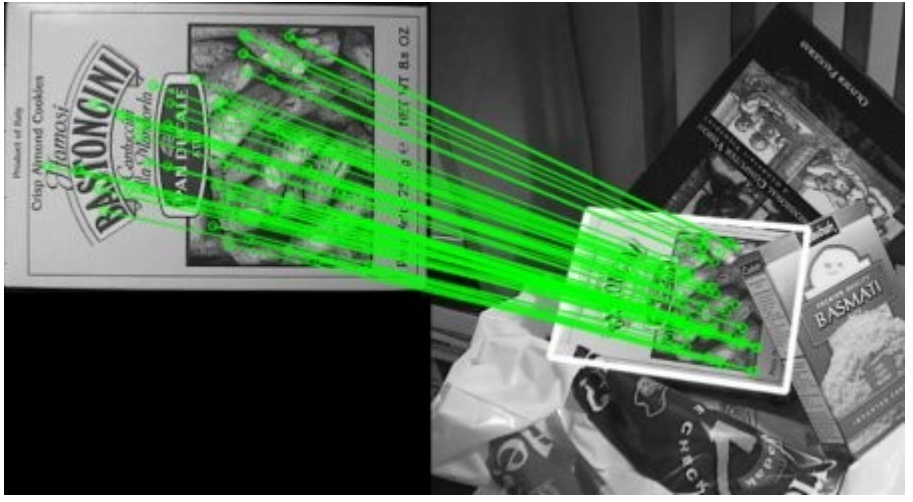
```
# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10], None, flags=2)

plt.imshow(img3),plt.show()
```



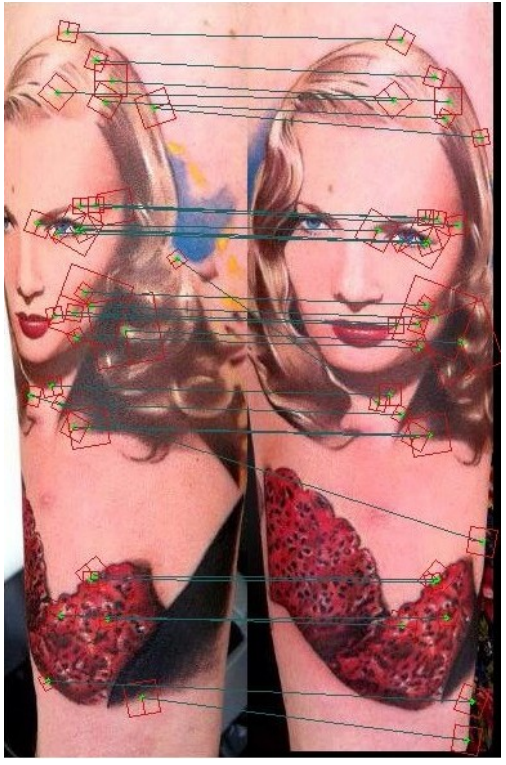
Desenhando os Matches

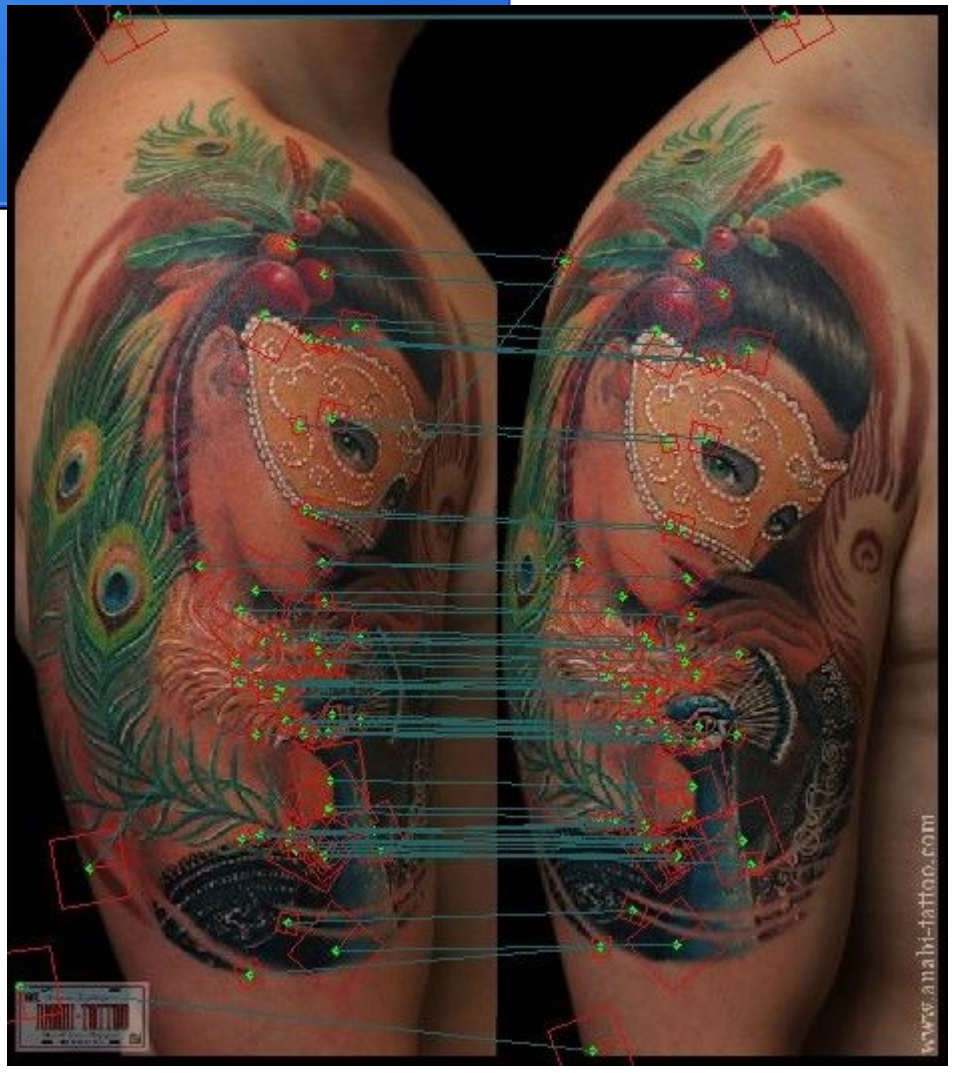
Teste a versão que está em:

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html



Exemplo de Aplicação





Tarefa

- O aluno deve fazer uma proposta de uma aplicação “real” que use matching de descritores.
- Fazer testes com os descritores SIFT, SURF, ORB, BRIEF e BRISK
- Deve criar uma pequena base de dados com pelo menos 20 imagens dessa aplicação. Esta base de dados deve estar variada de tal forma a se ter casos de sucesso e casos de fracasso.