IMAGE PROCESSING WITH SIFT

Assignment 5

Nicola Gugole

ID: 619625

Basic Keypoint Detection, Description

Fun Fact: Keypoint object has size field, but also response (strength), angle of orientation and pt for coordinates access

```python
def siftImg(img):
    sift = cv2.xfeatures2d.SIFT_create()
    #0 - get grayscale
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #1 - Detect and overlap to original img
    kp, des = sift.detectAndCompute(grayImg, None)
    resultImg = copy.deepcopy(img)
    cv2.drawKeypoints(grayImg,kp,resultImg,
        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return kp, des, resultImg
```

Drawing Keypoints with Correct Sizes

```python
fig, ax = plt.subplots(nrows=2, ncols=2)
for i in range(len(imgNames)):
    img = cv2.imread(os.getcwd()+"\\chosen_imgs\\"+imgNames[i])
    sift = cv2.xfeatures2d.SIFT_create()
    # get grayscale
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Detect
    kp, des = sift.detectAndCompute(grayImg, None)
    sorted_kp_des = sorted(zip(kp,des),
        key=lambda x : x[0].size, reverse=True) # order based on size
    kp = [x for x,_ in sorted_kp_des]
    des = [x for _,x in sorted_kp_des]
    to_draw = 0 # biggest keypoint
    cv2.drawKeypoints(grayImg,[kp[to_draw]],img,
        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=color)
    # preparing for plot
    ax[i][0].imshow(img)
    ax[i][1].bar(range(len(des[to_draw])),
        des[to_draw], color=bar_color[i])
fig.suptitle(f"Biggest size keypoint", fontsize=title_fontsize)
plt.show(block=False)
```

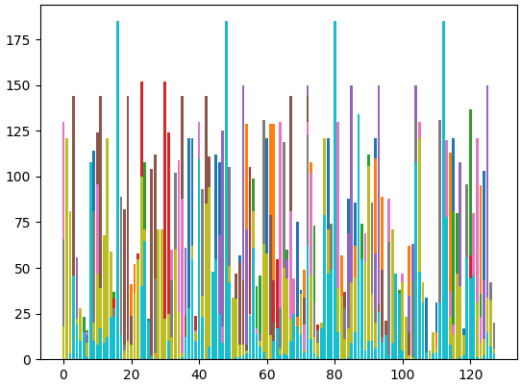
Biggest Keypoint Selection and Drawing for Visual Comparison

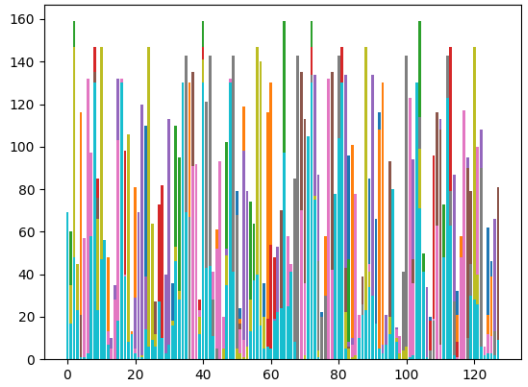
```python
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
fig, ax = plt.subplots(nrows=2, ncols=2)
to_print = matches[0] # print nearest keypoints between imgs
img4 = cv2.drawKeypoints(img1,[keypoints_1[to_print.queryIdx]],img1,
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(255,0,0))
img5 = cv2.drawKeypoints(img2,[keypoints_2[to_print.trainIdx]],img2,
    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(255,0,0))
ax[0][0].imshow(img4)
ax[0][1].bar(range(len(descriptors_1[to_print.queryIdx])),
    descriptors_1[to_print.queryIdx],
    color='r')
ax[1][0].imshow(img5)
ax[1][1].bar(range(len(descriptors_2[to_print.trainIdx])),
    descriptors_2[to_print.trainIdx],
    color='b')
fig.suptitle(f"Nearest Match\nEuclidean Distance: {round(to_print.distance,2)}")
plt.show()
```

Finding Matching Keypoints and ordering by Euclidean Distance

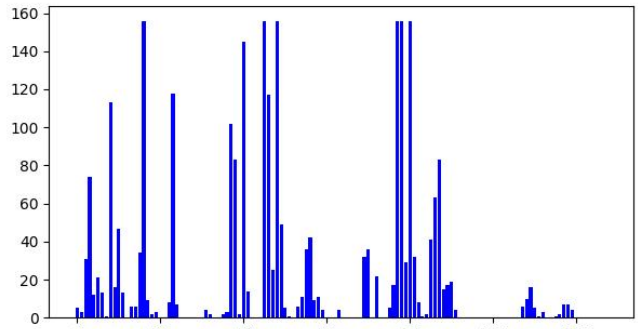
Drawing Nearest Match and Print Distance

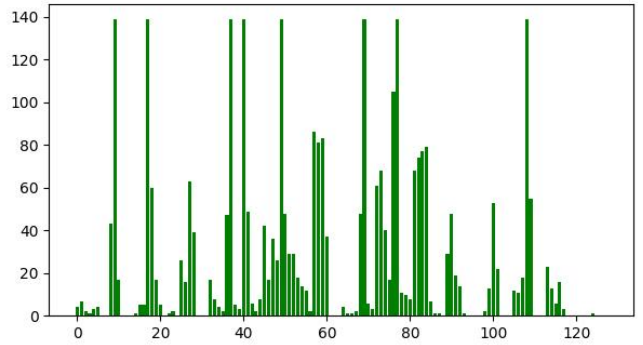
Keypoint Overlapping and Biggest KP Comparison



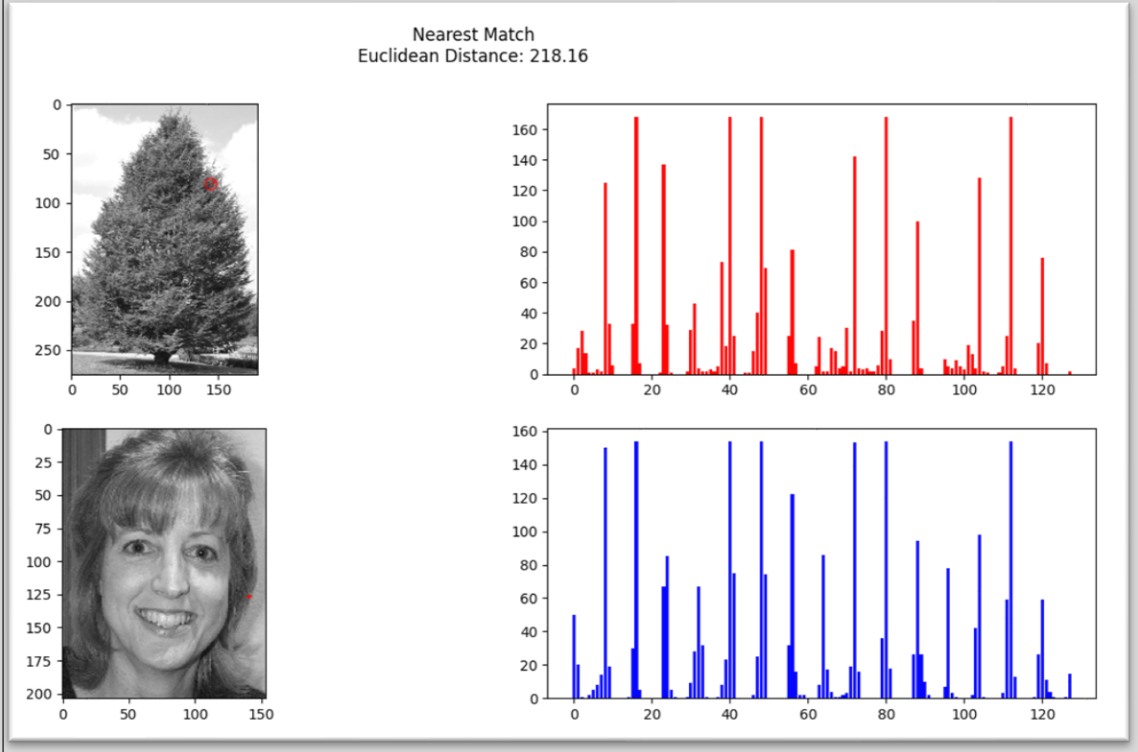


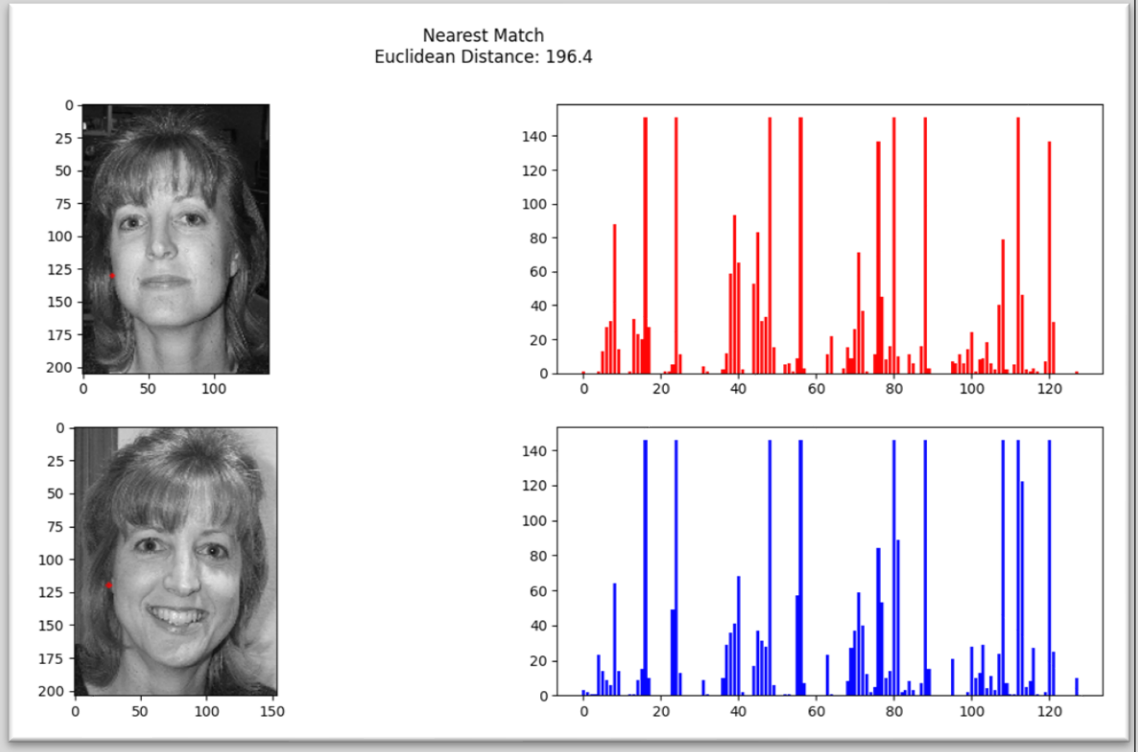
Biggest size keypoint





Matching Nearest Descriptors





Extremely different images may have incredibly similar descriptors..

Nearest descriptors in similar images make much more sense.
Single descriptors are NOT enough descriptive for the image, the set of descriptors are what makes the method work!

An application: Object Recognition

```python
# feature matching (Frame is image from video, Img is object to detect)
flannParam=dict(algorithm=0,tree=5)
matcher=cv2.FlannBasedMatcher(flannParam,{}) # faster than BF (bruteforce)

matches = matcher.knnMatch(descFrame,descImg, k=2) # take 2 nearest matches for single kp
goodMatches = []
minimumRate = 0.75
for m,n in matches:
    if m.distance < minimumRate * n.distance: # to check if match is "noisy" or not
        goodMatches.append(m)
if len(goodMatches)>=MIN_MATCHES: # good result, show matched img
    print("Found a match")
    resultImg = cv2.drawMatches(grayFrame, kpFrame, grayImg, kpImg, goodMatches,
        grayImg, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
else:
    print("Super sad, no match here, going on Tinder..")
    print(f"Found {len(goodMatches)}/{MIN_MATCHES}")
    resultImg = frame
return resultImg
```

Key Aspect

Take first and second nearest match for each keypoint, if distance of first one is not much smaller than second one:
BAD MATCH (not a good descriptor)
Find enough GOOD MATCHES and you have recognized the object

Quick Hints :

- Crop the image
- Use images with lot of details!

Object Recognition in Practice

THANK YOU FOR THE ATTENTION

(FOR CODE CURIOSITIES, HERE)