Computer Vision Lab 3

SIFT related exercises 2

In this lab, you will practice some of the implementation details that one may find relevant to SIFT (but we are not trying to implement SIFT in the labs as there are plenty of resources available in libraries or online). The lab only intends to get you familiarized with programming skills with Python and computer vision. We will try to computer dominant orientation and histogram of gradients of a given patch in this lab. We also make use of SIFT implementation of OpenCV to show SIFT points obtained of a given image.

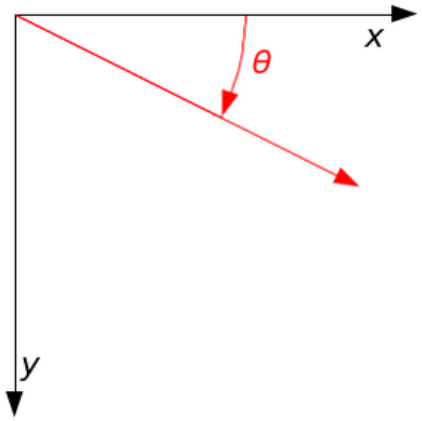
1. **Dominant orientation of a patch** The orientation of an image patch is determined by the eigen vector of the Hessian matrix computed in previous lab, corresponding to the larger eigen value. We can visualize the direction of an image patch centered at a given pixel, e.g. pixel at location (x=30, y=40), in the image by drawing a line indicating the direction. You can use OpenCV to draw a line using the function cv2.line(). For example, use the following code to draw a green line from point (x=30, y=40) to the point (x = 40, y =50) with thickness 1 onto an image img.

   ```
   cv2.line(img, (30, 40), (40, 50), (0, 255, 0), 1)
   ```

   Note that in Python, a matrix (image) is arranged in an row-major order. In other words, the first index is for the rows (y coordinates) and the second index is for the columns (x coordinates).

   In SIFT implementation, once we find the dominant direction of a patch, we suppose to rotate the patch window to align with the dominant direction and use the new patch for the subsequent descriptor computation. In this lab, to make things easier, we just visualize the rotated patch window by drawing it on the original image. Mathematically, to rotate a given point at x and y coordinate by angle θ (see the following figure), we should multiply the point with rotation

matrix $\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

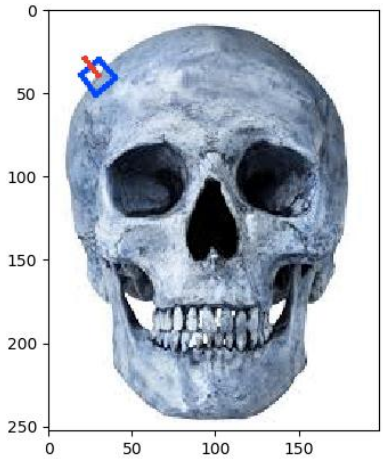


For a given patch, you can compute the u and v of the dominant direction of the patch and then $u/(u^2+v^2)^{0.5}$ and $v/(u^2+v^2)^{0.5}$ are the corresponding cos(θ) and sin(θ). Note θ is the angle between the positive direction of x-axis and the dominant direction. Because in Python the eigen vector is already scaled to unit length (the length of the vector is 1), we can use u for cos(θ) and v for sin(θ). Once having the rotation matrix, we can visualize the rotated patch window by rotating the four corners of the window and draw a square by connecting them with lines. For example,

```
sinTheta = directionV
cosTheta = directionU
# rotation matrix
R = np.array([[cosTheta, -sinTheta], [sinTheta, cosTheta]])
p1 = [-7, -7]
p2 = [-7, +7]
p3 = [+7, +7]
p4 = [+7, -7]
p1new = np.dot(R, p1) + [30, 40]
p2new = np.dot(R, p2) + [30, 40]
p3new = np.dot(R, p3) + [30, 40]
p4new = np.dot(R, p4) + [30, 40]
pts = np.array([p1new, p2new, p3new, p4new], np.int32)
pts.reshape((-1, 1, 2))
```

```
# draw the rotated window and dominant direction
img1 = img.copy()
img1 = cv2.polylines(img1, [pts], True, (0, 0, 255), 2,
cv2.LINE_AA)
```

For the point (x=30, y=40), your figure should look like the following.



2. **Key point descriptor** Next you will compute the histogram of gradients for a given patch. After computing the x and y gradients using Sobel filter for a patch, you can calculate the magnitude and orientation for each pixel. Then you can create a histogram of gradient orientation by dividing a full circle of orientation into 8 bins. You can use np.arctan2() to compute orientation given the x and y gradients as follows.

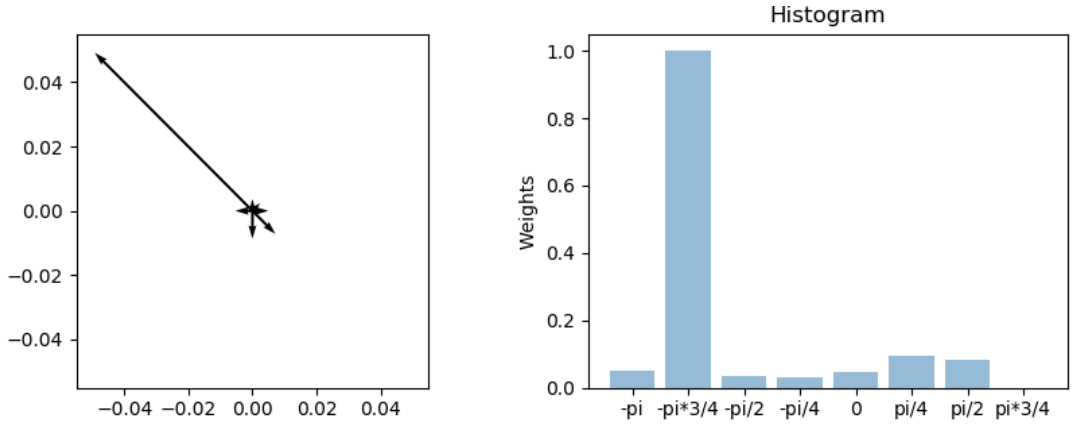
```
gradNorm = (gxPatch**2+gyPatch**2)**0.5
gradOrient = np.arctan2(gyPatch, gxPatch)
# divide the range [-pi,pi)to 8 segments
binsize = 2*np.pi/8
bin = np.arange(-np.pi, np.pi, binsize)
```

Note that gxPatch and gyPatch is the gradients for the patch and arctan2 will output the angle from -pi to pi (360 degree). For the weights in each bin, you then add the magnitudes of all pixels that has orientation in a given bin. Do this for every bin. Note that by using the above code, we actually try to make bins such as [-pi*7/8,-pi*5/8], [-pi*5/8, -pi*3/8]… You can use the following code to visualize the histogram as arrows and bar charts.

```
X = np.zeros(len(weights))
```

```
Y = X
scaledWeights = 1-np.exp(-weights)
plt.figure()
plt.subplot(1,2,1), plt.quiver(X, Y,
scaledWeights*np.cos(bin), -scaledWeights*np.sin(bin),
scale=1)
plt.axis('scaled')
# You can also view the weights as a barchart
objects = ('-pi','-pi*3/4','-pi/2','-
pi/4','0','pi/4','pi/2','pi*3/4','pi')
y_pos = np.arange(len(weights))
plt.subplot(1,2,2)
plt.bar(y_pos, weights, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Weights')
```

Note that weights are the histogram calculated and we scale them using $1\text{-}e^{-weights}$ for easier visualization (you do not have to do this). The figure should look like the following.

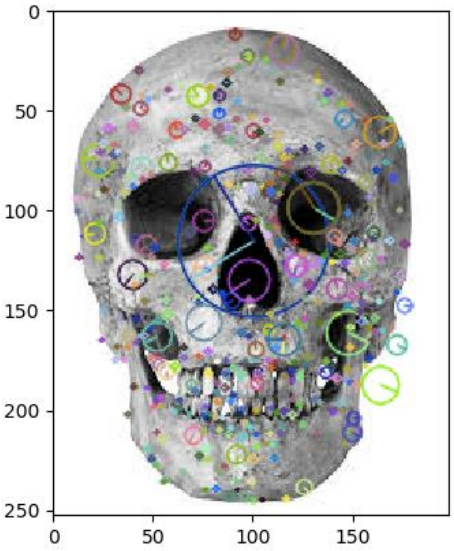


3. **OpenCV SIFT** Next you will try to use the SIFT implementation from OpenCV to see what the SIFT feature looks like. Use the following code.

```
sift = cv2.xfeatures2d.SIFT_create()
kp, des = sift.detectAndCompute(gray, None)
imgSift = img.copy()
imgSift=cv2.drawKeypoints(gray,kp,imgSift,flags=cv2.DRAW_MATCH
ES_FLAGS_DRAW_RICH_KEYPOINTS)
```

```
plt.figure()
plt.imshow(imgSift)
```

The figure should look like the following.



You can also look at des for what SIFT descriptor look like. For more details of SIFT in OpenCV, you can read

https://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html