03b_HOG

November 13, 2019

1 Assignment 3: Histogram of Oriented Gradients (HOG)

1.1 Read paper

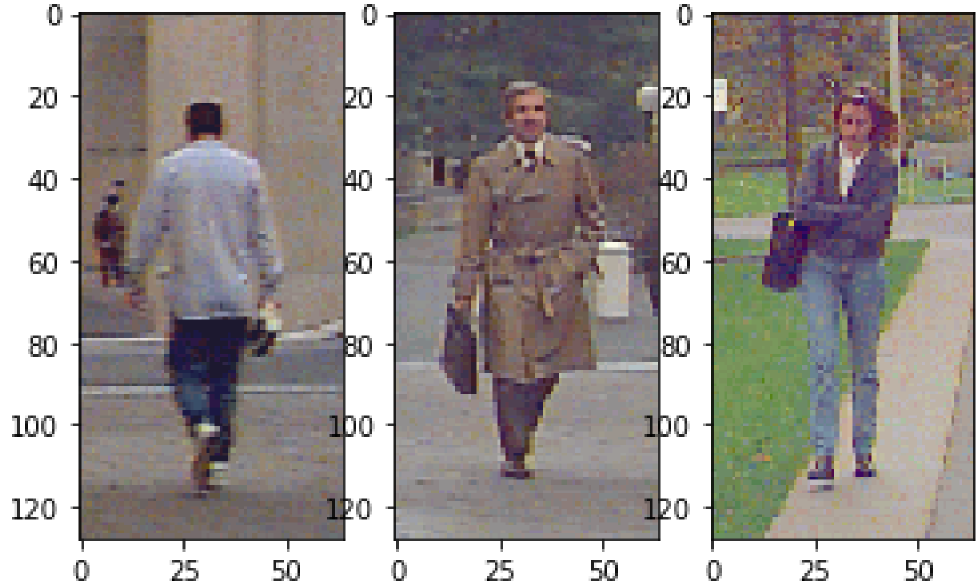
Have a look at the section "Resources" in the KVV and read the original HOG work of Dalal and Triggs from 2005 and the good summary by Tomasi (I would read this first). Now you shouldn't have any problems with the implementation of the algorithm!

1.2 MIT-DB for people

Download the complete image data set here: http://pascal.inrialpes.fr/data/human/

In [1]:
```
%matplotlib inline
from skimage import io,color, transform
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage
from scipy import spatial
import warnings
warnings.filterwarnings("ignore")

pers1 = io.imread('images/per00002.ppm')
pers2 = io.imread('images/per00007.ppm')
pers3 = io.imread('images/per00014.ppm')

fig = plt.figure()
ax1 = plt.subplot(1, 3, 1)
ax2 = plt.subplot(1, 3, 2)
ax3 = plt.subplot(1, 3, 3)

ax1.imshow(pers1)
ax2.imshow(pers2)
ax3.imshow(pers3)
```

Out[1]:
```
<matplotlib.image.AxesImage at 0x112c7f668>
```



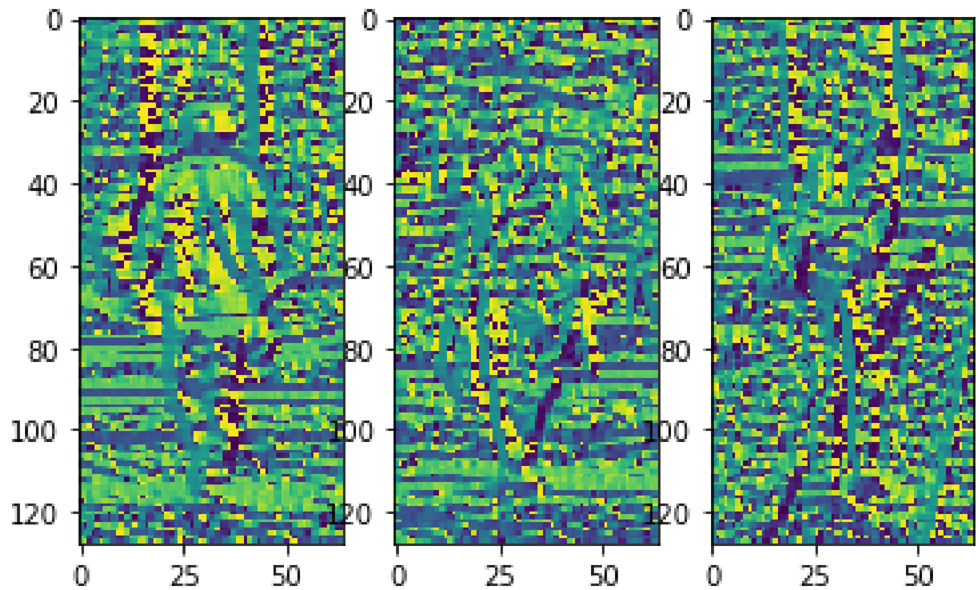
1.3 Gradients and directions

First, implement the extraction of the gradient via convolution with the Sobel kernels. Translate the two result matrices into an image containing the gradient direction (one angle per pixel). Display this image for each of the three input images above. Like it fancy? Add transparency inversely proportional to the gradient magnitude (weak gradients are transparent).

```
In [2]: # dieser Code ist Teil eine Musterlösung, die von Adrian Defer zur Verfügung gestellt


        def getGradientConv(image):
            Sx = np.array([
                [-1, 0, 1],
                [-1, 0, 1],
                [-1, 0, 1],
            ])
            Sy = Sx.T

            #gray for now TODO: Take max gradient of the 3 colorchannels as pixelvalue
            image = color.rgb2gray(image)
            #print(image.dtype)

            #image = np.float_(image)
            #print(image.dtype)

            Gx = ndimage.convolve(image, Sx)
            Gy = ndimage.convolve(image, Sy)
```

```
        '''
        #max G of color channels
        Gx0 = ndimage.convolve(image[:,:,0], Sx)
        Gy0 = ndimage.convolve(image[:,:,0], Sy)
        Gx1 = ndimage.convolve(image[:,:,1], Sx)
        Gy1 = ndimage.convolve(image[:,:,1], Sy)
        Gx2 = ndimage.convolve(image[:,:,2], Sx)
        Gy2 = ndimage.convolve(image[:,:,2], Sy)
        Gx = np.max(Gx0,Gx1,Gx2)'''

        #magnitude
        G = np.sqrt(np.square(Gx) + np.square(Gy))
        #direction
        angles = np.degrees(np.arctan2(Gy, Gx))

        return angles, G

    ang_pers1, _ = getGradientConv(pers1)
    ang_pers2, _ = getGradientConv(pers2)
    ang_pers3, _ = getGradientConv(pers3)

    fig = plt.figure()
    ax1 = plt.subplot(1, 3, 1)
    ax2 = plt.subplot(1, 3, 2)
    ax3 = plt.subplot(1, 3, 3)
    ax1.imshow(ang_pers1)
    ax2.imshow(ang_pers2)
    ax3.imshow(ang_pers3)
```

```
Out[2]: <matplotlib.image.AxesImage at 0x112e58f28>
```



1.4 Gradient orientation histograms (GOHs)

Now implement a function that generates GOHs on image cells. Pass a gradient direction image (as described above) to the function. The image cells should be 8 x 8 pixels in size. Please do not forget the voting with "bi-linear interpolation" (important!). Compute the GOH for the input image "star.png" with and without interpolation! Now rotate the image by -5ř and compute both variants of the GOH again! What are the Euclidean differences of the histograms (original vs rotated) with and without interpolation?

In [3]:
```
star = io.imread('images/star.png')
star_rot = ndimage.rotate(star, -5, mode='nearest')
#print(star_rot.shape)
#print(star.shape)
#plt.imshow(star)

cell_shape = 8

def hist_with_bilinear_binning(angles, gradients):
    bins = np.linspace(-180, 180, num=9)
    indices = np.digitize(angles, bins)

    indices_flat = indices.flatten()
    gra_flat = gradients.flatten()
    angles_flat = angles.flatten()

    hist = np.zeros((8), dtype=float)
```

```
    for i in range(len(gra_flat)):
        dist_left = abs(bins[(indices_flat[i]-1) % len(bins)] - angles_flat[i])
        dist_mid = abs(bins[(indices_flat[i]) % len(bins)] - angles_flat[i])
        dist_right = abs(bins[(indices_flat[i]+1) % len(bins)] - angles_flat[i])

        index = np.where(bins == angles_flat[i])

        #get the 2 min distances of the 3 distances to the 3 nearest indices
        if(dist_left > dist_mid):
            first_min = dist_mid
            hist[(index[0]) % 8] += gra_flat[i] * (first_min/45.0)

            if(dist_left > dist_right):
                second_min = dist_right
                hist[(index[0]+1) % 8] += gra_flat[i] * (second_min/45.0)
            else:
                second_min = dist_left
                hist[(index[0]-1) % 8] += gra_flat[i] * (second_min/45.0)
        else:
            first_min = dist_left
            hist[(index[0]-1) % 8] += gra_flat[i] * (first_min/45.0)

            if(dist_mid > dist_right):
                second_min = dist_right
                hist[(index[0]+1) % 8] += gra_flat[i] * (second_min/45.0)
            else:
                second_min = dist_mid
                hist[(index[0]) % 8] += gra_flat[i] * (second_min/45.0)

        #print(hist)

    return hist

def hist_wo_bilinear_binning(angles, gradients):

    bins = np.linspace(-180, 180, num=9)
    indices = np.digitize(angles, bins)
    theta = np.zeros(angles.shape)
    theta[:] = bins[(indices[:] - 1) % 8]

    gra_flat = gradients.flatten()
    theta_flat = theta.flatten()

    hist = np.zeros((8), dtype=float)

    for i in range(len(gra_flat)):
        index = np.where(bins == theta_flat[i])
        hist[index[0]] += gra_flat[i]
```

```python
            #print(hist)

        #hist = np.histogram(theta ,bins)
        return hist

    def GOH_simple(image):
        padding_x = cell_shape
        padding_y = cell_shape
        image_height = image.shape[0]
        image_width = image.shape[1]
        result = []

        for row in range(padding_y, image_height - padding_y, cell_shape):
            for col in range(padding_x, image_width - padding_x, cell_shape):
                cell = image[row:row+cell_shape, col:col+cell_shape]
                ang, gra = getGradientConv(cell)
                hist = hist_wo_bilinear_binning(ang, gra)
                result.append(hist)
        return result

    def GOH_bilinear(image):
        padding_x = cell_shape
        padding_y = cell_shape
        image_height = image.shape[0]
        image_width = image.shape[1]
        result = []

        for row in range(padding_y, image_height - padding_y, cell_shape):
            for col in range(padding_x, image_width - padding_x, cell_shape):
                cell = image[row:row+cell_shape, col:col+cell_shape]
                ang, gra = getGradientConv(cell)
                hist = hist_with_bilinear_binning(ang, gra)
                result.append(hist)
        return result

    a = GOH_simple(star)
    b = GOH_simple(star_rot)
    c = GOH_bilinear(star)
    d = GOH_bilinear(star_rot)

    #EUCLIDEAN DISTANCES
    # star and star_rot produce different count of histograms, due to different sizes. 399
    # Therefore, I cut off the cell hostogram count at 399 for now; the tendency is still
    # distance is way lower for bilinear interpolation binning
    '''
    print(len(a))
    print(len(b))
    print(len(c))
```

```
print(len(d))
'''

diff_simple = 0.0
diff_bilinear = 0.0

for j in range(len(a)):
    diff_simple += spatial.distance.euclidean(a[j],b[j])
    diff_bilinear += spatial.distance.euclidean(c[j],d[j])
diff_simple = diff_simple / len(a)
diff_bilinear = diff_simple / len(a)

print("Euclidean between star and star_rot W/O bilinear interpolation binning: " + str
print("Euclidean between star and star_rot WITH bilinear interpolation binning: " + st
```

```
Euclidean between star and star_rot W/O bilinear interpolation binning: 11.306048087500399
Euclidean between star and star_rot WITH bilinear interpolation binning: 0.02833596011904862
```

1.5 Block and ROI normalization and creation of the HOG descriptor

- Given a fixed size ROI (128 x 64 pixels), create a descriptor as shown in the lecture (including normalization of the blocks and the total ROI). Please reuse the precalculated cell histograms for the blocks!
- Calculate the descriptor for the three images above and for the unknown images "unknown1.png" and "unknown2.png".
- Now compare the unknowns with the three person instances and calculate the Euclidean distance of the descriptors. Which picture has a smaller distance to the persons? (**RESULT**)

```
In [ ]: # not enough time unfortunately :/
```