**CS 280 – Homework #2**

**Problem 1: Hybrid Images**

1. (a) (b) (c) Code: See Appendix 1
2. …
3. Algorithm: The algorithm is based on the facts that:

* The human eye can only see high frequencies from a close distance and cannot see high frequencies from a far distance.
* The human eye can only see low frequencies from a far distance and cannot see low frequencies from a close distance.

In brief, human eyes can see high frequencies from a close distance and can see low frequencies from a far distance. So, we low-pass filter image 1 and eliminate its high frequencies so that human eyes can see it from a far distance; we high-pass filter image 2 and eliminate its low frequencies so that human eyes can see it from a close distance. Subsequently, we linearly combine the filtered image 1 and the filtered image 2 to create a hybrid image from which human eyes can see the image 2 from a close distance and the image 1 from a far distance.

The parametric from of low-pass and high-pass filters are:

where for both Gaussian filters, the cutoff frequencies are 10 and 5 for image 1 and image 2, respectively (i.e. image 1 has frequencies less than or equal 10, image 2 has frequencies larger than or equal 5).

|  |  |
| --- | --- |
| G:\hw2_package\1_hybrid\gaussian_low_pass_filter.jpg | G:\hw2_package\1_hybrid\gaussian_high_pass_filter.jpg |
| Gaussian low-pass filter | Gaussian high-pass filter |

1. Favorite result:

|  |  |
| --- | --- |
| G:\hw2_package\1_hybrid\cat+dog_hybrid.jpg | G:\hw2_package\1_hybrid\cat+dog_hybrid.jpg |
| Dog from a close distance | Cat from a far distance |

Frequency analysis:

|  |  |
| --- | --- |
| G:\hw2_package\1_hybrid\cat.png Ground truth cat | G:\hw2_package\1_hybrid\dog.png Ground truth dog |
| G:\hw2_package\1_hybrid\cat+dog_im1_input_fft_log.jpg log of fft of input cat | G:\hw2_package\1_hybrid\cat+dog_im2_input_fft_log.jpg log of fft of input dog |
| G:\hw2_package\1_hybrid\cat+dog_im1_filtered.jpg cat filtered (low freqs) | G:\hw2_package\1_hybrid\cat+dog_im2_filtered.jpg dog filtered (high freqs) |
| G:\hw2_package\1_hybrid\cat+dog_im1_filtered_fft_log.jpg log of fft of filtered cat | G:\hw2_package\1_hybrid\cat+dog_im2_filtered_fft_log.jpg log of fft of filtered dog |
| G:\hw2_package\1_hybrid\cat+dog_hybrid.jpg hybrid cat + dog | G:\hw2_package\1_hybrid\cat+dog_hybrid_fft_log.jpg log of fft of hybrid cat+dog |

1. Two more results:

|  |  |
| --- | --- |
| G:\hw2_package\1_hybrid\malik.png  G:\hw2_package\1_hybrid\papadimitriou.png  Prof. Malik (above)  Prof. Papadimitriou (below) | G:\hw2_package\1_hybrid\malik+papadimitriou_hybrid.jpgG:\hw2_package\1_hybrid\malik+papadimitriou_hybrid.jpg Hybrid Malik+Papadimitriou at close distance (Papa) and at far distance (Malik) |
|  |  |
| G:\hw2_package\1_hybrid\boy.png  G:\hw2_package\1_hybrid\girl.png  Boy and girl | G:\hw2_package\1_hybrid\boy+girl_hybrid.jpg  G:\hw2_package\1_hybrid\boy+girl_hybrid.jpg Hybrid boy+girl at close distance (girl) and at far distance (boy) |

After a lot of trials and errors, I got some tricks to create a good hybrid:

* Cutoff frequencies: should be chosen so that the remaining information of both images should be relatively equal, i.e. image 1 is not too dim compared to image 2 and vice versa. Usually, the one without low frequencies is dimmer than the other, so we should choose its cutoff frequency not too large.
* Alignment: should match eyes and mouth, or pose.
* Linear combination: adjust the weight when adding up two images.
* Selection of images: they should have relatively similar size. Etc.

**Problem 2: Edge Detection**

1. **Finite operator**

Code [mag, theta] = difference filter(img): see Appendix 2

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_3096.jpg | G:\hw2_package\2_edge\bsds_3096_im_filtered_Dx.jpg | G:\hw2_package\2_edge\bsds_3096_im_filtered_Dy.jpg |
| G:\hw2_package\2_edge\bsds_3096_im_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_3096_im_gradient_orientation.jpg | 1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_101087.jpg | G:\hw2_package\2_edge\bsds_101087_im_filtered_Dx.jpg | G:\hw2_package\2_edge\bsds_101087_im_filtered_Dy.jpg |
| G:\hw2_package\2_edge\bsds_101087_im_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_101087_im_gradient_orientation.jpg | 1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_156065.jpg | G:\hw2_package\2_edge\bsds_156065_im_filtered_Dx.jpg | G:\hw2_package\2_edge\bsds_156065_im_filtered_Dy.jpg |
| G:\hw2_package\2_edge\bsds_156065_im_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_156065_im_gradient_orientation.jpg | 1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_253027.jpg | G:\hw2_package\2_edge\bsds_253027_im_filtered_Dx.jpg | G:\hw2_package\2_edge\bsds_253027_im_filtered_Dy.jpg |
| G:\hw2_package\2_edge\bsds_253027_im_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_253027_im_gradient_orientation.jpg | 1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_300091.jpg | G:\hw2_package\2_edge\bsds_300091_im_filtered_Dx.jpg | G:\hw2_package\2_edge\bsds_300091_im_filtered_Dy.jpg |
| G:\hw2_package\2_edge\bsds_300091_im_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_300091_im_gradient_orientation.jpg | 1. Ground truth 2. Filtered by Dx 3. Filtered by Dy 4. Grad magnitude 5. Grad orientation |

To compute the gradient orientation from x and y filter response: We call the response Rx and Ry, the gradient orientation is given by atan(Ry/Rx).

In python, we just simply write them in the complex form (Rx + 1j\*Ry) and call the function numpy.angle on this complex array.

1. **Derivative of Gausian**

Code [mag, theta] = derivative gaussian filter(img, sigma): See Appendix 2

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\1_hybrid\gaussian_low_pass_filter.jpg | G:\hw2_package\2_edge\derivative_gaussian_filter_Dx.jpg | G:\hw2_package\2_edge\derivative_gaussian_filter_Dy.jpg |
| Gaussian filter | Derivative over Dx | Derivative over Dy |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_3096.jpg | G:\hw2_package\2_edge\bsds_3096_im_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_3096_im_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_101087.jpg | G:\hw2_package\2_edge\bsds_101087_im_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_101087_im_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_156065.jpg | G:\hw2_package\2_edge\bsds_156065_im_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_156065_im_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_253027.jpg | G:\hw2_package\2_edge\bsds_253027_im_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_253027_im_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_300091.jpg | G:\hw2_package\2_edge\bsds_300091_im_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_300091_im_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

Observations: compared to the gradient magnitude and orientation of the simple derivative filters, the derivative Gaussian filter produces clearer edge boundary and orientation. The edges are thicker and the orientation has less noises. It is because the image is smoothened before the edges are detected.

It is because the convolution is the multiplication in frequency domain. The multiplication is commutative so the convolution is commutative in spatial domain. It is important because it allows us to apply as many Gaussian filters as we want (i.e. smoothing image many time) before we take the gradient. By this way, we can combine the Gaussian filters to save computational cost.

1. Oriented Filters

Code [mag,theta] = oriented filter(img): See Appendix 2

|  |  |
| --- | --- |
| G:\hw2_package\2_edge\oriented_gaussian_filter_Dx.jpg | G:\hw2_package\2_edge\derivative_oriented_gaussian_filter_Dx.jpg |
| G:\hw2_package\2_edge\oriented_gaussian_filter_Dy.jpg | G:\hw2_package\2_edge\derivative_oriented_gaussian_filter_Dy.jpg |
| G:\hw2_package\2_edge\oriented_gaussian_filter_Dxy.jpg | G:\hw2_package\2_edge\derivative_oriented_gaussian_filter_Dxy.jpg |
| G:\hw2_package\2_edge\oriented_gaussian_filter_Dyx.jpg | G:\hw2_package\2_edge\derivative_oriented_gaussian_filter_Dyx.jpg |
| Oriented Gaussian filters | Derivative of oriented Gaussian filters |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_3096.jpg | G:\hw2_package\2_edge\bsds_3096_im_oriented_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_3096_im_oriented_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_101087.jpg | G:\hw2_package\2_edge\bsds_101087_im_oriented_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_101087_im_oriented_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_156065.jpg | G:\hw2_package\2_edge\bsds_156065_im_oriented_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_156065_im_oriented_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_253027.jpg | G:\hw2_package\2_edge\bsds_253027_im_oriented_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_253027_im_oriented_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

|  |  |  |
| --- | --- | --- |
| G:\hw2_package\2_edge\bsds_300091.jpg | G:\hw2_package\2_edge\bsds_300091_im_oriented_gaussian_gradient_magnitude.jpg | G:\hw2_package\2_edge\bsds_300091_im_oriented_gaussian_gradient_orientation.jpg |
| Ground truth | Gradient magnitude | Gradient orientation |

Explanation of choosing filters: two elongated derivative Gaussian filters Dx and Dy to capture edges in horizontal and vertical directions, two elongated derivative Gaussian filters Dxy and Dyx to capture edges in forward and backward diagonal directions.

To combine the filter responses: we combine them in pairs, two perpendicular filters make a pair. We use L2-norm rule to combine a response pair into magnitude and use complex argument rule to combine the pair into orientation (see problem 2, question 1 answer). Subsequently, we add pairs up equally because we deliberately choose equally distributed directions.

1. Comparison

The result in 3 is close to the 2014 state-of-the-art result from Isola et al. It is not as sharp as the Isola’ but if we increase the number of filters in various direction, we can capture more edges and make our result approach the Isola’s.

The human annotations have clear boundary at different levels from few to many details.

Our best algorithm does well in detecting edges in various directions.

It does struggle with reducing noise and balancing between the sharpness of detected edge and the smoothening effect of filters.

Some challenges and difficulties with edge detection:

* Distinguish between noise and edge
* Some images have many details (ex: hair, grass, etc.)
* Smoothening makes losing information
* Computational time increases when we use many oriented filters
* Etc.

**Appendix 1**

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

from align\_image\_code import align\_images

import numpy as np

from scipy.signal import convolve2d

from skimage import color

def standard\_gaussian(X, Y, sigma):

return 1 / (2 \* np.pi \* sigma\*\*2) \*\* 0.5 \* np.exp(-(X\*\*2. + Y\*\*2.) / (2 \* sigma\*\*2))

def normalize(x):

return (x - np.min(x)) / (np.max(x) - np.min(x))

def hybrid\_image(im1\_aligned, im2\_aligned, sigma1, sigma2):

filter\_factor = 15

threshold\_1 = sigma1

threshold\_2= sigma2

std\_1 = 5

std\_2 = 5

x = np.arange(-filter\_factor, filter\_factor + 1)

y = np.arange(-filter\_factor, filter\_factor + 1)

X, Y = np.meshgrid(x, y)

low\_pass\_filter = standard\_gaussian(X, Y, std\_1)

low\_pass\_filter /= np.sum(low\_pass\_filter)

impulse\_filter = np.zeros((2 \* filter\_factor + 1, 2 \* filter\_factor + 1))

impulse\_filter[filter\_factor, filter\_factor] = 1

high\_pass\_filter = standard\_gaussian(X, Y, std\_2)

high\_pass\_filter /= np.sum(high\_pass\_filter)

high\_pass\_filter = impulse\_filter - high\_pass\_filter

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, low\_pass\_filter, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("gaussian\_low\_pass\_filter.jpg")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, high\_pass\_filter, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("gaussian\_high\_pass\_filter.jpg")

im1\_filtered = np.zeros(im1\_aligned.shape)

im2\_filtered = np.zeros(im2\_aligned.shape)

for i in range(3):

im1\_filtered[:, :, i] = convolve2d(im1\_aligned[:, :, i], low\_pass\_filter, mode="same")

im1\_filtered\_fft = np.fft.fft2(im1\_filtered[:, :, i])

# im1\_filtered\_fft[(threshold\_1 + 1):(-threshold\_1), (threshold\_1 + 1):(-threshold\_1)] = 0

im1\_filtered\_fft[(threshold\_1 + 1):(-threshold\_1), :] = 0

im1\_filtered\_fft[:, (threshold\_1 + 1):(-threshold\_1)] = 0

im1\_filtered[:, :, i] = np.real(np.fft.ifft2(im1\_filtered\_fft))

im2\_filtered[:, :, i] = convolve2d(im2\_aligned[:, :, i], high\_pass\_filter, mode="same")

im2\_filtered\_fft = np.fft.fft2(im2\_filtered[:, :, i])

# im2\_filtered\_fft[1:(threshold\_2+1), 1:(threshold\_2+1)] = 0

# im2\_filtered\_fft[-threshold\_2:, -threshold\_2:] = 0

im2\_filtered\_fft[1:(threshold\_2+1), :] = 0

im2\_filtered\_fft[-threshold\_2:, :] = 0

im2\_filtered\_fft[:, 1:(threshold\_2+1)] = 0

im2\_filtered\_fft[:, -threshold\_2:] = 0

im2\_filtered[:, :, i] = np.real(np.fft.ifft2(im2\_filtered\_fft))

im1\_filtered = normalize(im1\_filtered)

plt.imsave(common\_name + "im1\_filtered.jpg", im1\_filtered, format="jpg")

plt.figure()

plt.imshow(im1\_filtered)

plt.show()

im1\_filtered\_fft\_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im1\_filtered)))))

plt.imsave(common\_name + "im1\_filtered\_fft\_log.jpg", im1\_filtered\_fft\_log, format="jpg")

plt.imshow(im1\_filtered\_fft\_log)

plt.show()

im2\_filtered = normalize(im2\_filtered)

plt.imsave(common\_name + "im2\_filtered.jpg", im2\_filtered, format="jpg")

plt.imshow(im2\_filtered)

plt.show()

im2\_filtered\_fft\_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im2\_filtered)))))

plt.imsave(common\_name + "im2\_filtered\_fft\_log.jpg", im2\_filtered\_fft\_log, format="jpg")

plt.imshow(im2\_filtered\_fft\_log)

plt.show()

hybrid = 0.5 \* im1\_filtered + 0.5 \* im2\_filtered

return normalize(hybrid)

if \_\_name\_\_ == "\_\_main\_\_":

image\_pairs = [("malik.png", "papadimitriou.png"), ("cat.png", "dog.png"), ("boy.png", "girl.png")]

for im1\_name, im2\_name in image\_pairs:

common\_name = im1\_name.split('.')[0] + '+' + im2\_name.split('.')[0] + '\_'

# First load images

# high sf

im1 = plt.imread(im1\_name)

im1 = im1[:,:,:3]

im1\_input\_fft\_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im1)))))

plt.imsave(common\_name + "im1\_input\_fft\_log.jpg", im1\_input\_fft\_log, format="jpg")

plt.imshow(im1\_input\_fft\_log)

plt.show()

# low sf

im2 = plt.imread(im2\_name)

im2 = im2[:,:,:3]

im2\_input\_fft\_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(im2)))))

plt.imsave(common\_name + "im2\_input\_fft\_log.jpg", im2\_input\_fft\_log, format="jpg")

plt.imshow(im2\_input\_fft\_log)

plt.show()

# Next align images (this code is provided, but may be improved)

im1\_aligned, im2\_aligned = align\_images(im1, im2)

## You will provide the code below. Sigma1 and sigma2 are arbitrary

## cutoff values for the high and low frequencies

sigma1 = 10

sigma2 = 5

hybrid = hybrid\_image(im1\_aligned, im2\_aligned, sigma1, sigma2)

hybrid\_fft\_log = np.log(np.abs(np.fft.fftshift(np.fft.fft2(color.rgb2gray(hybrid)))))

plt.imsave(common\_name + "hybrid\_fft\_log.jpg", hybrid\_fft\_log, format="jpg")

plt.imshow(hybrid\_fft\_log)

plt.show()

plt.imsave(common\_name + "hybrid.jpg", hybrid, format="jpg")

plt.imshow(hybrid)

plt.show()

**Appendix 2**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

from scipy.signal import convolve2d

def normalize(x):

return (x - np.min(x)) / (np.max(x) - np.min(x))

def standard\_gaussian(X, Y, a, b, c, d, sigma\_1, sigma\_2):

return 1 / (np.pi \* (sigma\_1\*\*2 + sigma\_2\*\*2)) \*\* 0.5 \* np.exp( -(a \* X + b \* Y) \*\* 2. / (2 \* sigma\_1\*\*2) - (c \* X + d \* Y) \*\* 2. / (2 \* sigma\_2\*\*2))

def difference\_filter(I):

Dx = np.array([[1, -1]])

Dy = np.array([[1], [-1]])

I\_filtered\_Dx = np.zeros(I.shape)

I\_filtered\_Dy = np.zeros(I.shape)

for i in range(3):

I\_filtered\_Dx[:, :, i] = convolve2d(I[:, :, i], Dx, mode="same")

I\_filtered\_Dy[:, :, i] = convolve2d(I[:, :, i], Dy, mode="same")

return I\_filtered\_Dx, I\_filtered\_Dy

def derivative\_gaussian\_filter(I,sigma):

Dx = np.array([[1, -1]])

Dy = np.array([[1], [-1]])

filter\_factor = 15

x = np.arange(-filter\_factor, filter\_factor + 1)

y = np.arange(-filter\_factor, filter\_factor + 1)

X, Y = np.meshgrid(x, y)

gaussian\_filter = standard\_gaussian(X, Y, 1, 0, 0, 1, sigma, sigma)

gaussian\_filter\_Dx = convolve2d(gaussian\_filter, Dx, mode="same")

gaussian\_filter\_Dy = convolve2d(gaussian\_filter, Dy, mode="same")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_Dx, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("derivative\_gaussian\_filter\_Dx.jpg")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_Dy, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("derivative\_gaussian\_filter\_Dy.jpg")

I\_filtered\_Dx = np.zeros(I.shape)

I\_filtered\_Dy = np.zeros(I.shape)

for i in range(3):

I\_filtered\_Dx[:, :, i] = convolve2d(I[:, :, i], gaussian\_filter\_Dx, mode="same")

I\_filtered\_Dy[:, :, i] = convolve2d(I[:, :, i], gaussian\_filter\_Dy, mode="same")

return I\_filtered\_Dx, I\_filtered\_Dy

def oriented\_filter(I):

Dx = np.array([[1, -1]])

Dy = np.array([[1], [-1]])

Dxy = np.array([[0, 1], [-1, 0]])

Dyx = np.array([[1, 0], [0, -1]])

filter\_factor = 15

x = np.arange(-filter\_factor, filter\_factor + 1)

y = np.arange(-filter\_factor, filter\_factor + 1)

X, Y = np.meshgrid(x, y)

gaussian\_filter\_x = standard\_gaussian(X, Y, 1, 0, 0, 1, 2, 6)

gaussian\_filter\_Dx = convolve2d(gaussian\_filter\_x, Dx, mode="same")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_x, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("oriented\_gaussian\_filter\_Dx.jpg")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_Dx, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("derivative\_oriented\_gaussian\_filter\_Dx.jpg")

gaussian\_filter\_y = standard\_gaussian(X, Y, 1, 0, 0, 1, 6, 2)

gaussian\_filter\_Dy = convolve2d(gaussian\_filter\_y, Dy, mode="same")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_y, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("oriented\_gaussian\_filter\_Dy.jpg")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_Dy, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("derivative\_oriented\_gaussian\_filter\_Dy.jpg")

gaussian\_filter\_xy = standard\_gaussian(X, Y, 1, -1, 1, 1, 2, 6)

gaussian\_filter\_Dxy = convolve2d(gaussian\_filter\_xy, Dxy, mode="same")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_xy, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("oriented\_gaussian\_filter\_Dxy.jpg")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_Dxy, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("derivative\_oriented\_gaussian\_filter\_Dxy.jpg")

gaussian\_filter\_yx = standard\_gaussian(X, Y, 1, -1, 1, 1, 6, 2)

gaussian\_filter\_Dyx = convolve2d(gaussian\_filter\_yx, Dyx, mode="same")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_yx, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("oriented\_gaussian\_filter\_Dyx.jpg")

fig = plt.figure()

ax = fig.gca(projection='3d')

surf = ax.plot\_surface(X, Y, gaussian\_filter\_Dyx, rstride=1, cstride=1, cmap=cm.jet)

fig.colorbar(surf)

plt.savefig("derivative\_oriented\_gaussian\_filter\_Dyx.jpg")

I\_filtered\_Dx = np.zeros(I.shape)

I\_filtered\_Dy = np.zeros(I.shape)

I\_filtered\_Dxy = np.zeros(I.shape)

I\_filtered\_Dyx = np.zeros(I.shape)

for i in range(3):

I\_filtered\_Dx[:, :, i] = convolve2d(I[:, :, i], gaussian\_filter\_Dx, mode="same")

I\_filtered\_Dy[:, :, i] = convolve2d(I[:, :, i], gaussian\_filter\_Dy, mode="same")

I\_filtered\_Dxy[:, :, i] = convolve2d(I[:, :, i], gaussian\_filter\_Dxy, mode="same")

I\_filtered\_Dyx[:, :, i] = convolve2d(I[:, :, i], gaussian\_filter\_Dyx, mode="same")

return I\_filtered\_Dx, I\_filtered\_Dy, I\_filtered\_Dxy, I\_filtered\_Dyx

if \_\_name\_\_ == "\_\_main\_\_":

image\_name\_list = ["bsds\_3096.jpg", "bsds\_300091.jpg", "bsds\_253027.jpg", "bsds\_156065.jpg", "bsds\_101087.jpg"]

for image\_name in image\_name\_list:

name = image\_name.split('.')[0] + "\_"

im = plt.imread(image\_name)

# Simple derivative filters

im\_filtered\_Dx, im\_filtered\_Dy = difference\_filter(im)

im\_filtered\_Dx\_normalized = normalize(im\_filtered\_Dx)

plt.imsave(name + "im\_filtered\_Dx.jpg", im\_filtered\_Dx\_normalized, format="jpg")

plt.imshow(im\_filtered\_Dx\_normalized)

plt.show()

im\_filtered\_Dy\_normalized = normalize(im\_filtered\_Dy)

plt.imsave(name + "im\_filtered\_Dy.jpg", im\_filtered\_Dy\_normalized, format="jpg")

plt.imshow(im\_filtered\_Dy\_normalized)

plt.show()

im\_gradient\_magnitude = (im\_filtered\_Dx \*\* 2 + im\_filtered\_Dy \*\* 2) \*\* 0.5

im\_gradient\_magnitude\_2d = np.sum(im\_gradient\_magnitude, axis=2)

plt.imsave(name + "im\_gradient\_magnitude.jpg", im\_gradient\_magnitude\_2d, format="jpg", cmap="gray")

plt.imshow(im\_gradient\_magnitude\_2d, cmap="gray")

plt.show()

im\_gradient\_orientation = np.angle(im\_filtered\_Dx + 1j \* im\_filtered\_Dy)

im\_gradient\_orientation[np.where(im\_gradient\_magnitude < 10)] = 0

im\_gradient\_orientation\_2d = np.sum(im\_gradient\_orientation, axis=2)

plt.imsave(name + "im\_gradient\_orientation.jpg", im\_gradient\_orientation\_2d, format="jpg")

plt.imshow(im\_gradient\_orientation\_2d)

plt.show()

# Derivative Gaussian filters

im\_gaussian\_filtered\_Dx, im\_gaussian\_filtered\_Dy = derivative\_gaussian\_filter(im, 3)

im\_gaussian\_filtered\_Dx\_normalized = normalize(im\_gaussian\_filtered\_Dx)

plt.imsave(name + "im\_gaussian\_filtered\_Dx.jpg", im\_gaussian\_filtered\_Dx\_normalized, format="jpg")

plt.figure()

plt.imshow(im\_gaussian\_filtered\_Dx\_normalized)

plt.show()

im\_gaussian\_filtered\_Dy\_normalized = normalize(im\_gaussian\_filtered\_Dy)

plt.imsave(name + "im\_gaussian\_filtered\_Dy.jpg", im\_gaussian\_filtered\_Dy\_normalized, format="jpg")

plt.imshow(im\_gaussian\_filtered\_Dy\_normalized)

plt.show()

im\_gaussian\_gradient\_magnitude = (im\_gaussian\_filtered\_Dx \*\* 2 + im\_gaussian\_filtered\_Dy \*\* 2) \*\* 0.5

im\_gaussian\_gradient\_magnitude\_2d = np.sum(im\_gaussian\_gradient\_magnitude, axis=2)

plt.imsave(name + "im\_gaussian\_gradient\_magnitude.jpg", im\_gaussian\_gradient\_magnitude\_2d, cmap="gray")

plt.imshow(im\_gaussian\_gradient\_magnitude\_2d, cmap="gray")

plt.show()

im\_gaussian\_gradient\_orientation = np.angle(im\_gaussian\_filtered\_Dx + 1j \* im\_gaussian\_filtered\_Dy)

im\_gaussian\_gradient\_orientation[np.where(im\_gaussian\_gradient\_magnitude < 20)] = 0

im\_gaussian\_gradient\_orientation\_2d = np.sum(im\_gaussian\_gradient\_orientation, axis=2)

plt.imsave(name + "im\_gaussian\_gradient\_orientation.jpg", im\_gaussian\_gradient\_orientation\_2d)

plt.imshow(im\_gaussian\_gradient\_magnitude\_2d)

plt.show()

# Oriented Gaussian filters

im\_oriented\_gaussian\_filtered\_Dx, im\_oriented\_gaussian\_filtered\_Dy, im\_oriented\_gaussian\_filtered\_Dxy, im\_oriented\_gaussian\_filtered\_Dyx = oriented\_filter(im)

im\_oriented\_gaussian\_filtered\_Dx\_normalized = normalize(im\_oriented\_gaussian\_filtered\_Dx)

plt.imsave(name + "im\_oriented\_gaussian\_filtered\_Dx.jpg", im\_oriented\_gaussian\_filtered\_Dx\_normalized, format="jpg")

plt.figure()

plt.imshow(im\_oriented\_gaussian\_filtered\_Dx\_normalized)

plt.show()

im\_oriented\_gaussian\_filtered\_Dy\_normalized = normalize(im\_oriented\_gaussian\_filtered\_Dy)

plt.imsave(name + "im\_oriented\_gaussian\_filtered\_Dy.jpg", im\_oriented\_gaussian\_filtered\_Dy\_normalized, format="jpg")

plt.imshow(im\_oriented\_gaussian\_filtered\_Dy\_normalized)

plt.show()

im\_oriented\_gaussian\_filtered\_Dxy\_normalized = normalize(im\_oriented\_gaussian\_filtered\_Dxy)

plt.imsave(name + "im\_oriented\_gaussian\_filtered\_Dxy.jpg", im\_oriented\_gaussian\_filtered\_Dxy\_normalized, format="jpg")

plt.imshow(im\_oriented\_gaussian\_filtered\_Dxy\_normalized)

plt.show()

im\_oriented\_gaussian\_filtered\_Dyx\_normalized = normalize(im\_oriented\_gaussian\_filtered\_Dyx)

plt.imsave(name + "im\_oriented\_gaussian\_filtered\_Dyx.jpg", im\_oriented\_gaussian\_filtered\_Dyx\_normalized, format="jpg")

plt.imshow(im\_oriented\_gaussian\_filtered\_Dyx\_normalized)

plt.show()

im\_oriented\_gaussian\_gradient\_magnitude = (im\_oriented\_gaussian\_filtered\_Dx \*\* 2 + im\_oriented\_gaussian\_filtered\_Dy \*\* 2

+ im\_oriented\_gaussian\_filtered\_Dxy \*\* 2 + im\_oriented\_gaussian\_filtered\_Dyx \*\* 2) \*\* 0.5

im\_oriented\_gaussian\_gradient\_magnitude\_2d = np.sum(im\_oriented\_gaussian\_gradient\_magnitude, axis=2)

plt.imsave(name + "im\_oriented\_gaussian\_gradient\_magnitude.jpg", im\_oriented\_gaussian\_gradient\_magnitude\_2d, format="jpg", cmap="gray")

plt.imshow(im\_oriented\_gaussian\_gradient\_magnitude\_2d, cmap="gray")

plt.show()

im\_oriented\_gaussian\_gradient\_orientation = np.angle(im\_oriented\_gaussian\_filtered\_Dx + 1j \* im\_oriented\_gaussian\_filtered\_Dy) \

+ np.angle(im\_oriented\_gaussian\_filtered\_Dxy + 1j \* im\_oriented\_gaussian\_filtered\_Dyx)

im\_oriented\_gaussian\_gradient\_orientation[np.where(im\_oriented\_gaussian\_gradient\_magnitude < 20)] = 0

im\_oriented\_gaussian\_gradient\_orientation\_2d = np.sum(im\_oriented\_gaussian\_gradient\_orientation, axis=2)

plt.imsave(name + "im\_oriented\_gaussian\_gradient\_orientation.jpg", im\_oriented\_gaussian\_gradient\_orientation\_2d, format="jpg")

plt.imshow(im\_oriented\_gaussian\_gradient\_orientation\_2d)

plt.show()