

# Lab Instructions - session 5

## Reading from camera device, edge detection

### Part 1. Reading from a webcam

The following code reads a video stream from a webcam and displays it.

File: `read_webcam.py`

```
import numpy as np
import cv2

cam_id = 0 # camera id

# for default webcam, cam_id is usually 0
# try out other numbers (1,2,..) if this does not work

cap = cv2.VideoCapture(cam_id)

while True:
    ret, I = cap.read();

    cv2.imshow("my stream", I);

    # press "q" to quit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

## Part 2. Edge detection

### Computing the Gradients

We compute Sobel gradients both by applying a Sobel filter and by using the OpenCV function "Sobel".

File: **sobel1.py**

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

I = cv2.imread("agha-bozorg.jpg", cv2.IMREAD_GRAYSCALE)

# Compute the gradient in x direction using the sobel filter

# Method 1: using filter2D *****
Dx = np.array([[ -1,  0,  1],
               [-2,  0,  2],
               [-1,  0,  1]]); # Sobel filter

Ix = cv2.filter2D(I, -1, Dx);
print(I.dtype)
print(Ix.dtype)

Ix = cv2.filter2D(I, cv2.CV_16S, Dx); # cv2.CV_16S: 16 bit signed integer
print(Ix.dtype)
raw_input('press any key to continue... ')

# Method 2: using sobel function *****
Ix2 = cv2.Sobel(I, cv2.CV_16S, 1, 0)
print(np.abs(Ix - Ix2).max())
raw_input('press any key to continue... ')

# Plot the gradient image
f, axes = plt.subplots(2, 2)

axes[0,0].imshow(I, cmap = 'gray')
axes[0,0].set_title("Original Image")

axes[0,1].imshow(Ix, cmap = 'gray')
axes[0,1].set_title("Ix (cv2.filter2D)")

axes[1,0].imshow(Ix2, cmap = 'gray')
axes[1,0].set_title("Ix2 (cv2.Sobel)")

axes[1,1].imshow(np.abs(Ix), cmap = 'gray')
axes[1,1].set_title("abs(Ix)")

# Notice that imshow in matplotlib considers the minimums value of I
# as black and the maximum value as white (this is different from
# the behavior in cv2.imshow
plt.show()
```

- Why have we used `cv2.CV_16S` (16 bit signed integer) format for the output of `filter2D` and `Sobel` functions, instead of `-1` (which gives the same numeric type as the input image, i.e. `CV_8U` or unsigned 8-bit integer)?
- What is the value of `np.abs(Ix - Ix2).max()`? Are `Ix` and `Ix2` different?
- Why are most of the pixels in images of `axes[0,1]` and `axes[1,0]` gray? What do the black and white pixels show in these images? (notice that `matplotlib` automatically sets the minimum value of an image to black and the maximum value to white)

## Towards edge detection!

The strength of the edge at each certain pixel can be represented by the magnitude of the gradient vector, that is

$$\sqrt{(\partial I / \partial x)^2 + (\partial I / \partial y)^2}$$

The next code plots the gradient in x and y directions, plus the magnitude of the gradient.

File: **sobel2.py**

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

I = cv2.imread("aha-bozorg.jpg", cv2.IMREAD_GRAYSCALE)

# Sobel gradient in the x direction
Ix = cv2.Sobel(I, cv2.CV_64F, 1, 0)
print(Ix.dtype)

# Sobel gradient in the y direction
Iy = cv2.Sobel(I, cv2.CV_64F, 0, 1)
print(Iy.dtype)

# Magnitude of gradient
E = np.sqrt(Ix*Ix + Iy*Iy)

# Plot the gradient image
f, axes = plt.subplots(2, 2)

axes[0,0].imshow(I, cmap = 'gray')
axes[0,0].set_title("Original Image")

axes[0,1].imshow(abs(Ix), cmap = 'gray')
axes[0,1].set_title("abs(Ix)")

axes[1,0].imshow(abs(Iy), cmap = 'gray')
axes[1,0].set_title("abs(Iy)")

axes[1,1].imshow(E, cmap = 'gray')
axes[1,1].set_title("Magnitude of Gradient")
plt.show()
```

## Edge detection (Sobel, Laplacian, Canny):

Now, we move on to edge detection. We use three different methods:

1. Using Sobel gradients + thresholding
2. Laplacian of Gaussian (LoG) + detection zero-crossings
3. Canny Edge detection

The function **zero\_crossing** finds zero crossings in an image for LoG edge detection. You do not need to know how the functions **std\_filter** and **zero\_crossing** work.

File: **edge.py**

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

def std_filter(I, ksize):
    F = np.ones((ksize,ksize), dtype=np.float) / (ksize*ksize);

    MI = cv2.filter2D(I,-1,F) # apply mean filter on I

    I2 = I * I; # I squared
    MI2 = cv2.filter2D(I2,-1,F) # apply mean filter on I2

    return np.sqrt(MI2 - MI * MI)

def zero_crossing(I):
    """Finds locations at which zero-crossing occurs, used for
    Laplacian edge detector"""

    Ishrx = I.copy();
    Ishrx[:,1:] = Ishrx[:, :-1]

    Ishdy = I.copy();
    Ishdy[1:, :] = Ishdy[:, :-1]

    ZC = (I==0) | (I * Ishrx < 0) | (I * Ishdy < 0); # zero crossing
    locations

    SI = std_filter(I, 3) / I.max()

    Mask = ZC & (SI > .1)

    E = Mask.astype(np.uint8) * 255 # the edges

    return E

I = cv2.imread("agha-bozorg.jpg", cv2.IMREAD_GRAYSCALE)

# set the sigma for Gaussian Blurring
sigma = 7

# Sobel magnitude of gradient
thresh = 90 # threshold
```

```
Ib = cv2.GaussianBlur(I, (sigma,sigma), 0); # blur the image
Ix = cv2.Sobel(Ib,cv2.CV_64F,1,0)
Iy = cv2.Sobel(Ib,cv2.CV_64F,0,1)
Es = np.sqrt(Ix*Ix + Iy*Iy)
Es = np.uint8(Es > thresh)*255 # threshold the gradients

# Laplacian of Gaussian
# Here, we first apply a Gaussian filter and then apply
# the Laplacian operator (instead of applying the LoG filter)
Ib = cv2.GaussianBlur(I, (sigma,sigma), 0);
E1 = cv2.Laplacian(Ib,cv2.CV_64F,ksize=5)
E1 = zero_crossing(E1);

# Canny Edge detector
lth = 50 # low threshold
hth = 120 # high threshold
Ib = cv2.GaussianBlur(I, (sigma,sigma), 0); # blur the image
Ec = cv2.Canny(Ib,lth, hth)
f, axes = plt.subplots(2, 2)

axes[0,0].imshow(I,cmap = 'gray')
axes[0,0].set_title("Original Image")

axes[0,1].imshow(Es,cmap = 'gray')
axes[0,1].set_title("Sobel")

axes[1,0].imshow(E1,cmap = 'gray')
axes[1,0].set_title("Laplacian")

axes[1,1].imshow(Ec,cmap = 'gray')
axes[1,1].set_title("Canny")

# Notice that imshow in matplotlib considers the minimums value of I
# as black and the maximum value as white (this is different from
# the behavior in cv2.imshow)
plt.show()
```

- Compare the Canny edge detector to Sobel+thresholding. Can you see the effect of non-maximum suppression?
- Notice that in all cases we first smooth the image using a Gaussian filter. Why is the purpose of smoothing the image. Change the smoothing parameter **sigma** (from **sigma=7** to **sigma=5, 3, 1** or **9**) and see what happens by increasing or decreasing this parameter.
- Change the low and high thresholds of the Canny edge detector and observe the results.

## Today's task:

You need to read a video stream from your webcam and apply different gradients or edge detection operations to the stream. You have to do this by completing the file `lab5_task1.py`. Your program must have the following functionalities:

- **press the 'o' key:** show the original webcam frame (already done)
- **press the 'x' key:** show the Sobel gradient in the x direction (already done)
- **press the 'y' key:** show the Sobel gradient in the y direction
- **press the 'm' key:** show the Sobel magnitude of the gradient
- **press the 's' key:** show the result of Sobel + thresholding edge detection
- **press the 'l' key:** apply Laplacian of Gaussian (LoG) edge detector
- **press the 'c' key:** apply Canny edge detector
- **press the '+' key:** increase smoothing parameter sigma (already done)
- **press the '-' key:** decrease smoothing parameter sigma (already done)
- **press the 'q' key:** quit the program (already done)

Notice that after reading each image frame we apply a Gaussian filter to blur it. All gradient/edge detection operations must be done on the blurred image `Ib`.

File: `lab5_task1.py`

```
import numpy as np
import cv2

cam_id = 0 # camera id

# for default webcam, cam_id is usually 0
# try out other numbers (1,2,..) if this does not work

cap = cv2.VideoCapture(cam_id)

mode = 'o' # show the original image at the beginning

sigma = 5

while True:
    ret, I = cap.read();
    #I = cv2.imread("agha-bozorg.jpg") # can use this for testing
    I = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY) # convert to grayscale
    Ib = cv2.GaussianBlur(I, (sigma,sigma), 0); # blur the image

    if mode == 'o':
        # J = the original image
        J = I
    elif mode == 'x':
        # J = Sobel gradient in x direction
        J = np.abs(cv2.Sobel(Ib,cv2.CV_64F,1,0));
```

```
elif mode == 'y':
    # J = Sobel gradient in y direction
    pass
elif mode == 'm':
    # J = magnitude of Sobel gradient
    pass
elif mode == 's':
    # J = Sobel + thresholding edge detection
    pass
elif mode == 'l':
    # J = Laplacian edges
    pass
elif mode == 'c':
    # J = Canny edges
    pass

# we set the image type to float and the maximum value to 1
# (for a better illustration) notice that imshow in opencv does not
# automatically map the min and max values to black and white.
J = J.astype(np.float) / J.max();
cv2.imshow("my stream", J);

key = chr(cv2.waitKey(1) & 0xFF)

if key in ['o', 'x', 'y', 'm', 's', 'c', 'l']:
    mode = key
if key == '-' and sigma > 1:
    sigma -= 2
    print("sigma = %d"%sigma)
if key in ['+', '=']:
    sigma += 2
    print("sigma = %d"%sigma)
elif key == 'q':
    break

cap.release()
cv2.destroyAllWindows()
```

- Wave your hand quickly. What happens to the edges? Why?
- Press the 's' and 'c' keys to compare **Canny** with **Sobel** edge detection. Explain the effect of non-maximum suppression to the TA.
- In each case, increase and decrease the **sigma** by pressing + and - keys and see what happens. Explain the reason to the TA.

## References

- [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_gradients/py\\_gradients.html#gradients](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_gradients/py_gradients.html#gradients)
- [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_canny/py\\_canny.html#canny](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_canny/py_canny.html#canny)