Handson 3

CIE5141 Computer Vision in Construction

b07501113 林庭瑄

助教好帥～（先加**50**分）

1.1 Import the required packages and load the input image

```
import cv2
from google.colab.patches import cv2_imshow
from google.colab import drive
drive.mount('/content/gdrive/')

# input image
!gdown --id '1cxbUexTPahDqof7Vy2l-4IKXFwybxs-s' --output
rebar_cage.jpg
# TODO: use opencv to read and show the image
path ="./rebar_cage.jpg"
originalImage = cv2.imread(path)
cv2_imshow(originalImage)
```

1.2 Convert to Grayscale

```
# TODO: use opencv to convert to the graysclae image
grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY )
cv2_imshow(grayImage)
```

2.1 Padding

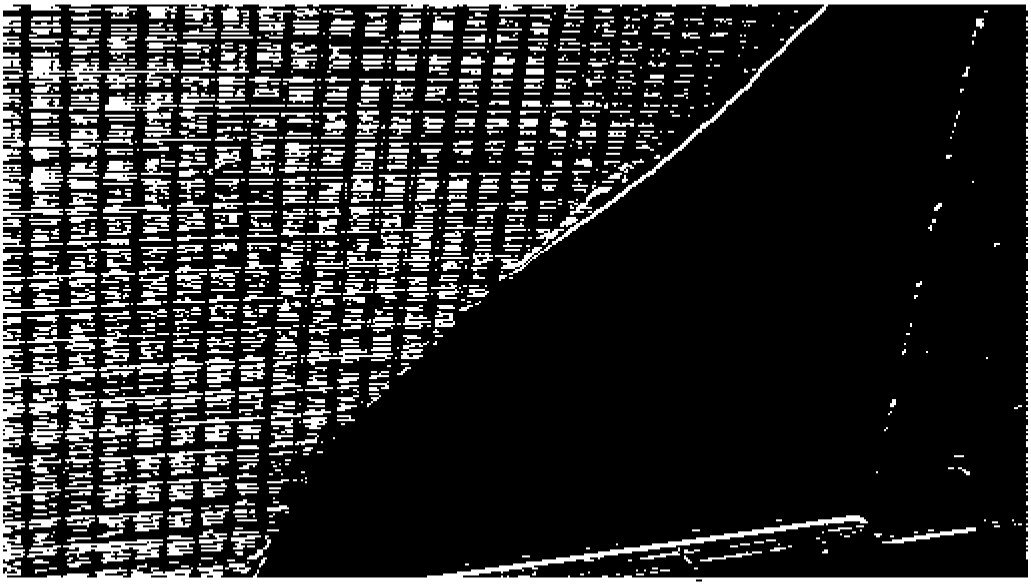
```
import numpy as np
# TODO: make a numpy array by using np.arange
arr1=np.arange(0,12)
arr1=np.reshape(arr1, (-1, 4))
print(arr1)
# TODO: use np.pad to apply padding with zeros arround the array.
# hint: np.pad([input],[padding_width],'constant')
arr2=np.pad(arr1, ((1,1),(1,1)), 'constant' ,constant_values=0)
print(arr2)
```

2.2 Sobel operator

2.2.1 Sobel y_component

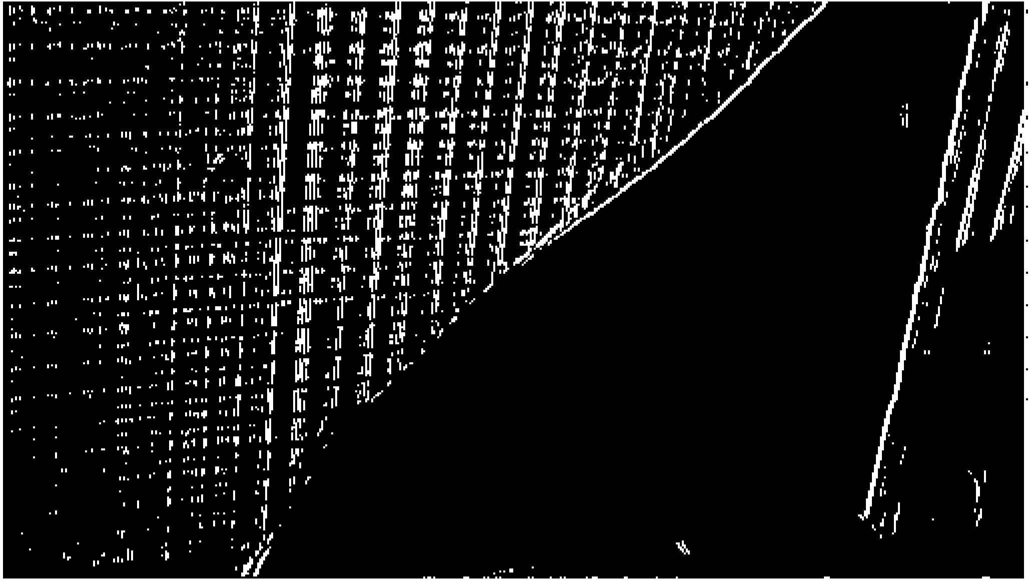
```
img=grayImage
def sobel_y_operator (input, threshold):
 # create a numpy zeros array whose shape should be same as
the input image
```

```
 img_y = np.zeros(input.shape)
 # TODO: create a sobel y-component filter
 # hint: sobel y-component filter is a 3x3 matric which is
mensioned above
 sobel_y = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])

 # TODO: Pad the input image with zeros. Padding width should
be equal half of the filter width.
 input_padding = np.pad(input, 1, 'constant'
,constant_values=0)

 # apply the Sobel filter. Avoid processing outside the
boundary
 for row in
range(int(sobel_y.shape[0]/2),input_padding.shape[0]-int(sobel
_y.shape[0]/2)):
   # TODO: set the range when processing the image in column
   for col in
range(int(sobel_y.shape[1]/2),input_padding.shape[1]-int(sobel
_y.shape[1]/2)):
     # TODO: compute the gradient in y direction of each pixel
     # hint: np.sum() can help you to sum up the elements, be
care of the index
     Gy =
np.sum(np.array(input_padding[row-1:row+2,col-1:col+2])*np.arr
ay(sobel_y[0:3,0:3]))
     # TODO: compute G, which is equals to sqrt(Gy**2+Gy**2)
     G=(Gy**2+Gy**2)**0.5
     # TODO: compare G with the threshold, if G > threshold,
turn the pixel white, else turn it black
     if G>threshold:
       img_y[row-2,col-2]=255
 return img_y
cv2_imshow(sobel_y_operator(img, 180))
```



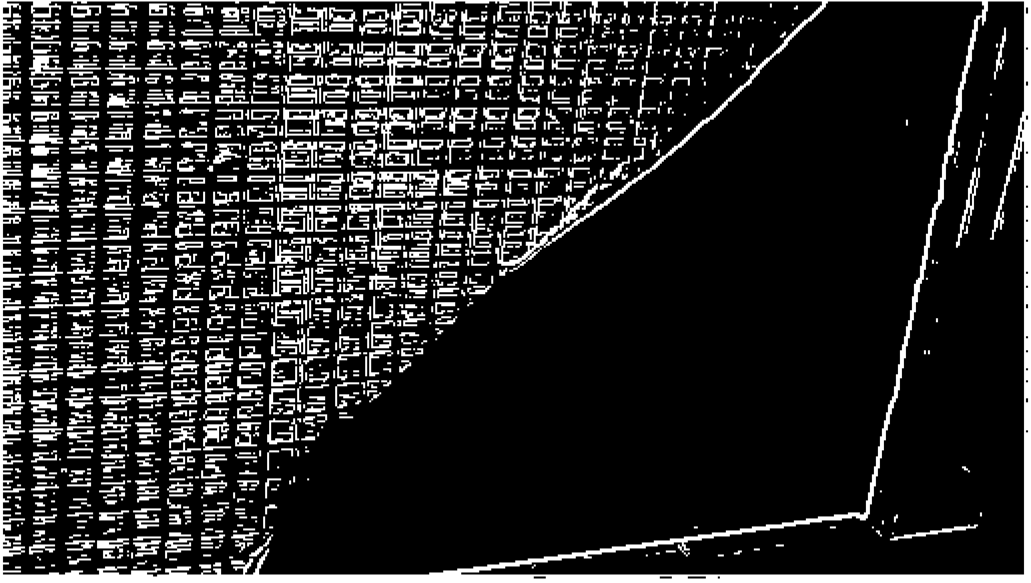
2.2.2 Sobel x_component

```
img=grayImage
def sobel_x_operator (input, threshold):
 # create a numpy zeros array whose shape should be same as
the input image
 img_x = np.zeros(input.shape)
 # TODO: create a sobel y-component filter
 # hint: sobel y-component filter is a 3x3 matric which is
mensioned above
 sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])

 # TODO: Pad the input image with zeros. Padding width should
be equal half of the filter width.
 input_padding = np.pad(input, 1, 'constant'
,constant_values=0)

 # apply the Sobel filter. Avoid processing outside the
boundary
 for row in
range(int(sobel_x.shape[0]/2),input_padding.shape[0]-int(sobel
_x.shape[0]/2)):
   # TODO: set the range when processing the image in column
   for col in
range(int(sobel_x.shape[1]/2),input_padding.shape[1]-int(sobel
_x.shape[1]/2)):
     # TODO: compute the gradient in y direction of each pixel
     # hint: np.sum() can help you to sum up the elements, be care of the index
     Gx = np.sum(np.array(input_padding[row-1:row+2,col-1:col+2])*np.array(sobel_x[0:3,0:3]))
     # TODO: compute G, which is equals to sqrt(Gy**2+Gy**2)
     G=(Gx**2+Gx**2)**0.5
     # TODO: compare G with the threshold, if G > threshold, turn the pixel white, else turn it black
     if G>threshold:
       img_x[row-2,col-2]=255
 return img_x
cv2_imshow(sobel_x_operator(img, 180))
```



2.2.3 Sobel

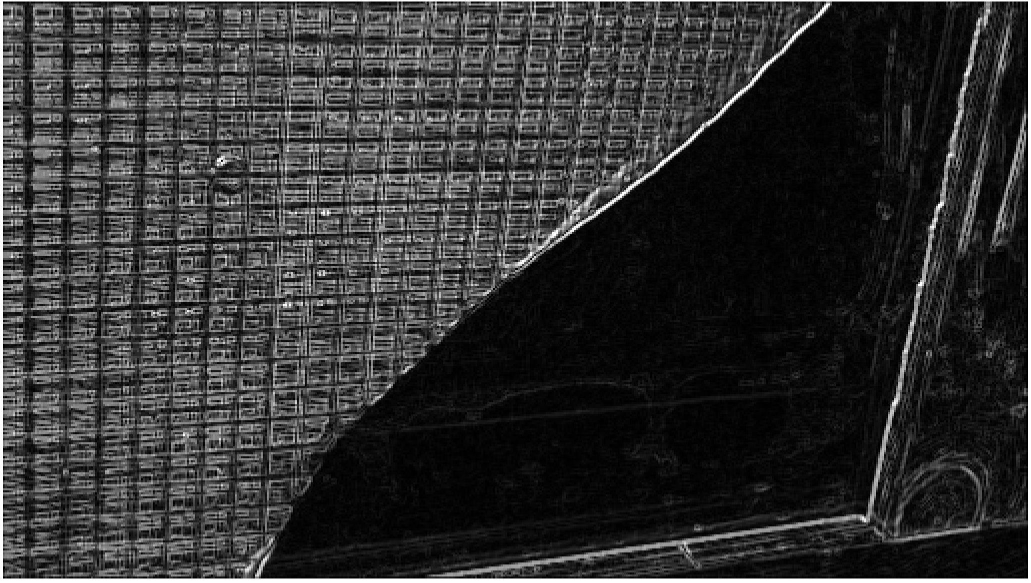
```
img=grayImage
# TODO: finish the sobel_operator function
def sobel_operator (input, threshold):
 # create a numpy zeros array whose shape should be same as the input image
 img_sobel = np.zeros(input.shape)
 sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
 sobel_y = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])
 # apply padding same as earlier
 input_padding = np.pad(input, 1, 'constant' ,constant_values=0)
 # apply the Sobel filter. Avoid processing outside the boundary
 for row in range(int(sobel_x.shape[0]/2),input_padding.shape[0]-int(sobel_x.shape[0]/2)):
   for col in range(int(sobel_x.shape[1]/2),input_padding.shape[1]-int(sobel_x.shape[1]/2)):
     Gx = np.sum(np.array(input_padding[row-1:row+2,col-1:col+2])*np.array(sobel_x[0:3,0:3]))
     Gy = np.sum(np.array(input_padding[row-1:row+2,col-1:col+2])*np.array(sobel_y[0:3,0:3]))

     # compute G, which is equals to sqrt(Gx**2+Gy**2)
     G=(Gx**2+Gy**2)**0.5
     # compare G with the threshold, if G > threshold, turn the pixel white, else turn it black
     if G>threshold:
       img_sobel[row-2,col-2]=255
 return img_sobel
cv2_imshow(sobel_operator(img, 180))
```



2.3 Edge detection with OpenCV

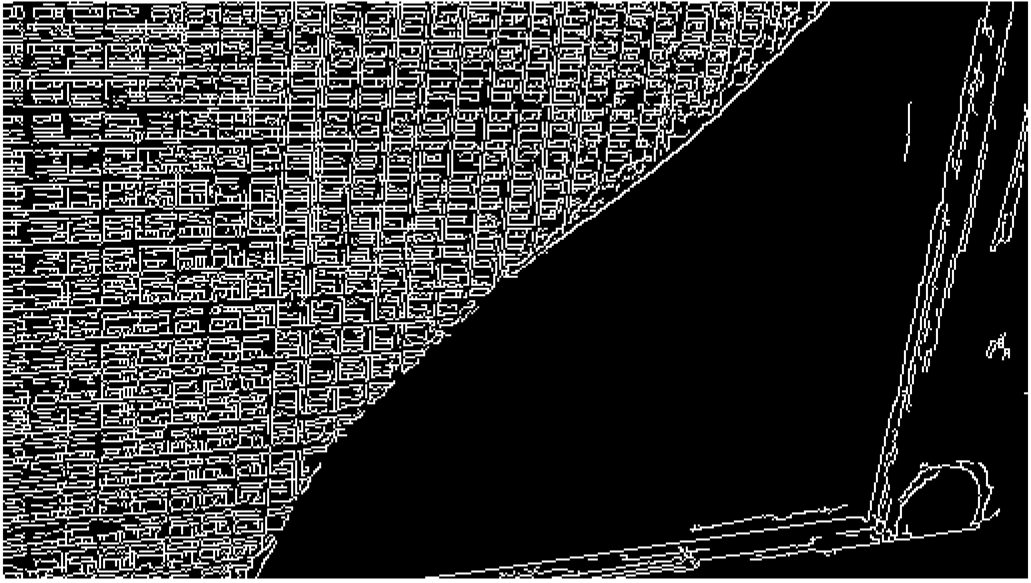
2.3.1 Sobel with OpenCV

```
# TODO: use cv2.Sobel() to calculate the derivatives from the image horizontally and vertically,
#      calculate the gradient at each point,
#      and compare the gradient with a proper threshold(you can decide it as long as the output is clear).
# hint: cv2.Sobel() can only calculate the derivatives
x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
y = cv2.Sobel(img, cv2.CV_16S, 0, 1)

absX = cv2.convertScaleAbs(x)# 轉回uint8
absY = cv2.convertScaleAbs(y)

dst = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)
cv2_imshow(dst)
```



2.3.2 Canny with OpenCV

```
# TODO: use cv2.Canny() to apply edge detection.
canny = cv2.Canny(img, 60, 215)
cv2_imshow(canny)
```



Bonus

3.3 Image processing with Gaussian filter

```
img=grayImage
def gaussian_operator (input):
 # create a numpy zeros array whose shape should be same as the
input image
 img_x = np.zeros(input.shape)
 # TODO: create a sobel y-component filter
 gaussian = make_5x5_gaussian_filter(1)

 # TODO: Pad the input image with zeros. Padding width should be
equal half of the filter width.
 input_padding = np.pad(input, int(gaussian.shape[0]/2),
'constant' ,constant_values=0)

 # apply the Sobel filter. Avoid processing outside the boundary
 for row in
range(int(gaussian.shape[0]/2),input_padding.shape[0]-int(gaussia
n.shape[0]/2)):
   # TODO: set the range when processing the image in column
   for col in
range(int(gaussian.shape[1]/2),input_padding.shape[1]-int(gaussia
n.shape[1]/2)):
     # TODO: compute the gradient in y direction of each pixel
     # hint: np.sum() can help you to sum up the elements, be
care of the index
     img_x[row-2,col-2]=
np.sum(np.array(input_padding[row-2:row+3,col-2:col+3])*np.array(
gaussian[0:5,0:5]))
 return img_x
cv2_imshow(gaussian_operator (img))
```