Lab 3: **Feature detection and matching**

Learning Outcomes:

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- Understand the knowledge of features detectors, features descriptors, edge detectors, and line detectors.
- Write a demo program, implements algorithms: Harris Corner Detector, Histogram of Oriented Gradients, Canny Operator, Hough Transform.

Requirements:

In this assignment, students are asked to write a program that implements algorithms for features detection, features description, edge detection, and line detection, window detection. Details of the functions are described below:

Function 1: Harris Corner Detector is the algorithm used for the feature detector. You are required to implement a Harris corner detector to perform feature detection

In [2]:

```
def find_harris_corners(input_img, k=0.04, window_size = 5, threshold = 1000):
    '''
    input_img: the image that we look for conners
    k: constant, deault 0.04
    window_size: size of window that we consider to look for conners, default 5x5
    threshold: to decide whether a pixel is a conner, default 0.01
    '''


    corners = []
    output_img = np.copy(input_img)
    if(len(output_img.shape) == 3): processed_img = cv2.cvtColor(output_img, cv2.COLOR_BGR2GRAY)
    print("output_img shape = ", output_img.shape)

    offset = int(window_size/2)
    y_range = processed_img.shape[0] - offset
    x_range = processed_img.shape[1] - offset

    # Any kernel (Sobel, ...) can be used to calculate gradient of input image
    dy, dx = np.gradient(processed_img)
    # print(dy)

    Ixx = dx**2
    Ixy = dy*dx
    Iyy = dy**2


    # your implementation here .....


    return corners, output_img
```

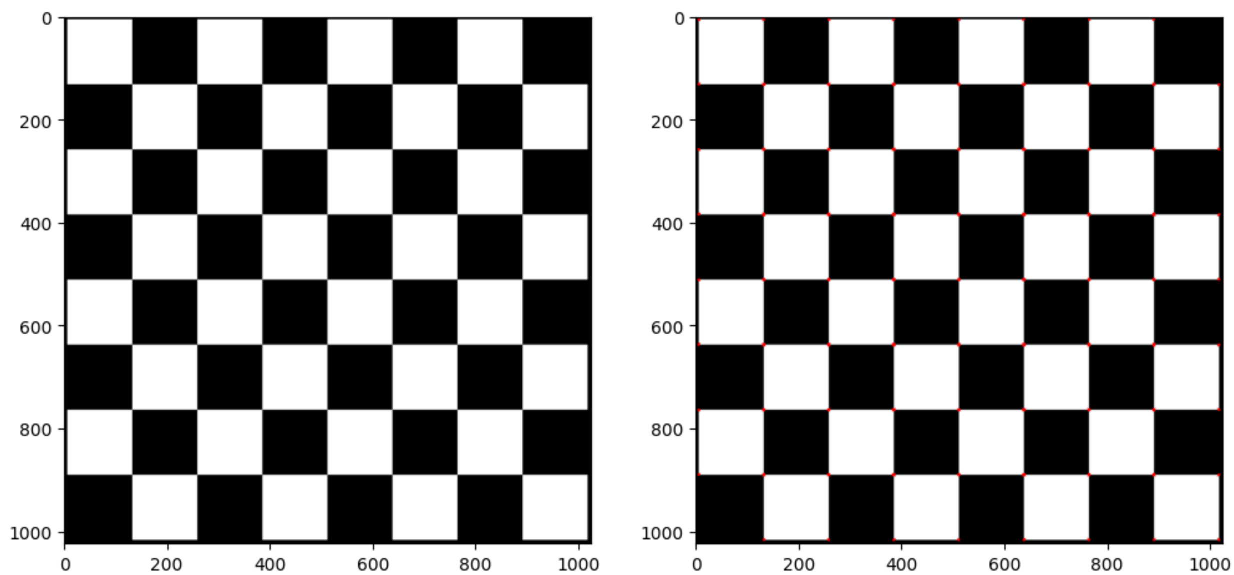
```
In [3]:
img = cv2.imread('chess.png') # , cv2.IMREAD_GRAYSCALE, chess.png
assert img is not None, "file could not be read, check with os.path.exists()"
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
In [4]:
print("img shape = ", img.shape)
# plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
img shape =  (449, 640, 3)
```

```
In [5]:
connerList, outputImg = find_harris_corners(img, k=0.06, window_size = 5, threshold = 40000.00)
# threshold: empirically selected
f, axis_array = plt.subplots(1,2, figsize=(12,8))
axis_array[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
axis_array[1].imshow(cv2.cvtColor(outputImg, cv2.COLOR_BGR2RGB))
```

```
output_img shape =  (449, 640, 3)
Out[5]: <matplotlib.image.AxesImage at 0x799d7970ca30>
```



Function 2: HOG is a histogram of orientations of the image gradients within a patch. The Histogram of Oriented Gradients method (or HOG for short) is used for object detection and image recognition. HOG is based on feature descriptors, which extract useful information and discard the unnecessary parts. HOG calculates the horizontal and vertical components of the gradient's magnitude and direction of each individual pixel and then organizes the information into a 9-bin histogram to determine shifts in the data. You are required to implement HOG in python to perform the feature description.

```
img = cv2.imread('popularImage.png') # , cv2.IMREAD_GRAYSCALE, chess.png
assert img is not None, "file could not be read, check with os.path.exists()"
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)[3:307, 3:307]
print(img_gray.shape)
```

```
(304, 304)
```

```
'''
Compute gradient of an image by rows and columns
'''
dx, dy = np.gradient(img_gray)
magnitude_gradient = np.sqrt(dy**2 + dx**2)
orientation_gradient = abs(np.rad2deg(np.arctan2(dy,dx))) # *180 / np.pi
# print(orientation_gradient)
```

```
bins = np.arange(0, 180, 20)
print(bins)
```

```
[  0  20  40  60  80 100 120 140 160]
```

```
def hog_cell(cell_mag, cell_dir, bins = [0, 20, 40, 60, 80, 100, 120, 140, 160]):
    '''
    magnitude of each orientation is distibuted to bins using interpolation algorithm
    '''
    cell_hist = np.zeros(len(bins))
    last_bin_idx = len(bins) - 1
    mags = cell_mag.flatten()
    dirs = cell_dir.flatten()
    # print(mags)
    # print(dirs)
    for i in range(len(dirs)):
        for b in range(len(bins)):
            # print(b)
            if(dirs[i] > bins[b] and b == len(bins)-1):
                cell_hist[8] += mags[i] - (dirs[i]-bins[last_bin_idx])/(180 - bins[last_bin_idx])*mags[i]
                cell_hist[0] += (dirs[i]-bins[last_bin_idx])/(180 - bins[last_bin_idx])*mags[i]
                # print("over 140", cell_hist)
            else:
                # print(Bins[b+1])
                if(dirs[i]>bins[b] and dirs[i]<=bins[b+1]):
                    cell_hist[b] += (bins[b+1]-dirs[i])/(bins[b+1] - bins[b])*mags[i]
                    cell_hist[b+1] += (dirs[i]-bins[b])/(bins[b+1] - bins[b])*mags[i]

    # print(cell_hist)
    return cell_hist


```

```
# for testing
cell_mag = np.array([[1, 2], [3, 4]])
cell_dir = np.array([[-125, 85], [150, 4]])
# print(cell)
cell_hist = hog_cell(cell_mag, cell_dir)
print(cell_hist, type(cell_hist))
```

```
[3.2 0.8 0.  0.  1.5 0.5 0.  1.5 1.5] <class 'numpy.ndarray'>
```

```python
def hist_of_oriented_grandient(img_gray, window_size = 16, cell_size = 4):
    '''
    In this example, window size (patch size) and cell size are given as default
    If lecturer want to change these value, remember to resize image to suitable size
    '''
    dy, dx = np.gradient(img_gray)
    # print(dy.shape)
    magnitude_gradient = np.sqrt(dy**2 + dx**2)
    # print(magnitude_gradient.shape)
    orientation_gradient = abs(np.rad2deg(np.arctan2(dy,dx))) # *180 / np.pi
    yRange = img_gray.shape[0] // window_size
    xRange = img_gray.shape[1] // window_size
    # print("yRange, xRange = ", yRange, xRange)

    img_hist = np.array([])
    for y in range(yRange):
        for x in range(xRange):
            window = img_gray[(y*window_size):((y+1)*window_size), (x*window_size):((x+1)*window_size)]
            window_magnitude_gradient = magnitude_gradient[(y*window_size):((y+1)*window_size), (x*window_size):
            window_orientation_gradient = orientation_gradient[(y*window_size):((y+1)*window_size), (x*window_si
            # if(y==yRange-1 and x==xRange-1): print(window_orientation_gradient.shape)
            window_hist = np.array([])
            iRange = int(np.sqrt(window_size/cell_size))
            for i in range(iRange):
                for j in range(iRange):
                    # if(y==0 and x==0): print("i, j = ", i, j)
                    cell_mag = window_magnitude_gradient[(i*cell_size):((i+1)*cell_size), (j*cell_size):((j+1)*c
                    cell_dir = window_orientation_gradient[(i*cell_size):((i+1)*cell_size), (j*cell_size):((j+1)
                    cell_hist = hog_cell(cell_mag, cell_dir)
                    # if(y==0 and x==0): print("cell_hist.shape = ", cell_hist.shape)
                    window_hist = np.hstack((window_hist,cell_hist))
                    # print(window_hist)
                    # if(y==0 and x==0): print("cell_hist shape, window_hist.shape = ", cell_hist.shape, window_
            img_hist = np.hstack((img_hist,window_hist))

    return img_hist.reshape(yRange*xRange, 36)
```

```python
hog_fratures = hist_of_oriented_grandient(img_gray)
# print(hog_fratures)
print("hog_fratures = ", hog_fratures.shape)
```

Function 3: Edges are significant local changes of intensity in a digital image. An edge can be defined as a set of connected pixels that forms a boundary between two disjoint regions. You are required to implement the Canny Operator in python to perform edge detection.

Solution (please follow the theory slides).

```
canny_img = Canny_detector(img_gray, 50, 150)
```

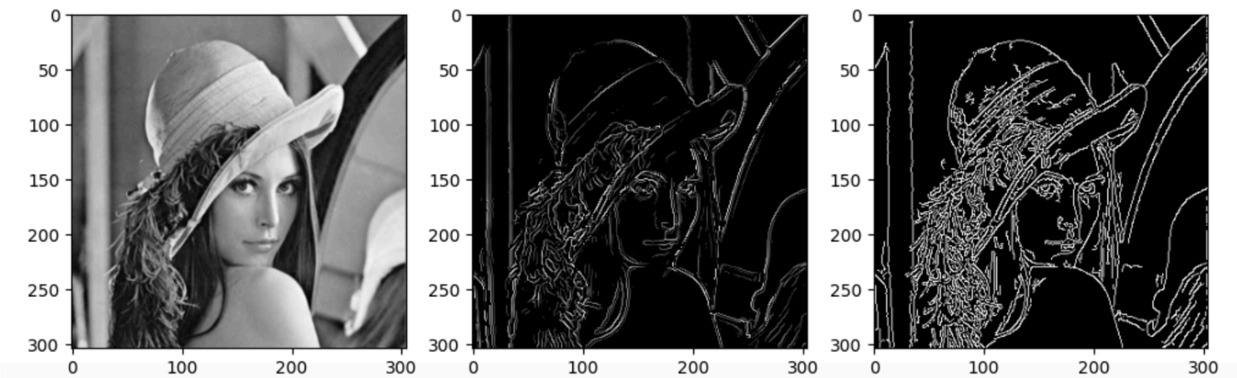
```
edges = cv2.Canny(img_gray, 50, 150, apertureSize=3)
```

```
print(canny_img.shape)
print(edges.shape)
```

```
(304, 304)
(304, 304)
```

```
f, axis_array = plt.subplots(1,3, figsize=(12,4))
axis_array[0].imshow(cv2.cvtColor(img_gray, cv2.COLOR_GRAY2RGB))
axis_array[1].imshow(cv2.cvtColor(canny_img, cv2.COLOR_GRAY2RGB))
axis_array[2].imshow(cv2.cvtColor(edges, cv2.COLOR_GRAY2RGB))
```

```
<matplotlib.image.AxesImage at 0x799d329e4280>
```



Function 4: Hough transform is used to recognize complex lines in photographs. For the Hough Transform algorithm, it is crucial to perform edge detection first to produce an edge image which will then be used as input into the algorithm. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. You are required to implement Hough transform in python to perform line detection(Rectangle detection)

```python
def houghLine(image):
  #Get image dimensions
  # y for rows and x for columns
  Ny = image.shape[0]
  Nx = image.shape[1]

  #Max diatance is diagonal one
  Maxdist = int(np.round(np.sqrt(Nx**2 + Ny ** 2)))

  # Theta in range from -90 to 90 degrees
  thetas = np.deg2rad(np.arange(-90, 90))
  #Range of radius
  rs = np.linspace(-Maxdist, Maxdist, 2*Maxdist)
  accumulator = np.zeros((2 * Maxdist, len(thetas)))

  for y in range(Ny):
    for x in range(Nx):
      # Check if it is an edge pixel
      #  NB: y -> rows , x -> columns
      if image[y,x] > 0:
        # Map edge pixel to hough space
        for k in range(len(thetas)):
          # Calculate space parameter
          r = x*np.cos(thetas[k]) + y * np.sin(thetas[k])
          # Update the accumulator
          # N.B: r has value -max to max
          # map r to its idx 0 : 2*max
          accumulator[int(r) + Maxdist,k] += 1
  return accumulator, thetas, rs
```

```python
image = np.zeros((150,150))
image[75, 75] = 1
image[50, 50] = 1
```

```python
accumulator, thetas, rhos = houghLine(image)
f, axarr = plt.subplots(1,2, figsize = (10, 8))
axarr[0].title.set_text('Original Image')
axarr[0].imshow(image, cmap = 'Greys')
# plt.set_cmap('gray')
axarr[1].title.set_text('Hough Space')
axarr[1].imshow(accumulator, cmap = 'Greys')
# plt.set_cmap('gray')
# plt.show()
```

Evaluation Criteria

No	Criteria	Requires	Mark	Note
1	Function 1: Harris Corner Detector	Implement Harris Corner Detector	2	Using mouse or keyboard
2	Function 2: Histogram of Oriented Gradients	Implment Histogram of Oriented Gradients	3	Using mouse or keyboard
3	Function 3: Canny Operator for edge detection	Implement Canny Operator to detect edge	2	Using mouse or keyboard

4	Function 4: Hough transform	Implement Hough transform to detect the windows	4	Using mouse or keyboard
5	Total		10	