Importing Modules

In [1]:
```
import numpy as np
```

Defining Methods

In [4]:

```
# Getting sliding window
#img is the image and m is the size of window (eg 3x3, 5x5)
def sliding_window(img,kernel):

         #make the loop for 3 and 5.
    s = img.shape[0]
    for i in range(int(kernel/2)):
        z = np.zeros([s,1])
        img = np.concatenate((z,img),axis=1)

        z = np.zeros([s,1])
        img = np.concatenate((img,z),axis=1)

        s = s+2

        z = np.zeros([1,s])
        img = np.concatenate((img,z),axis=0)

        z = np.zeros([1,s])
        img = np.concatenate((z,img),axis=0)


    window= []
    stepSize = 1
    w_width = kernel
    w_height = w_width
    for x in range(0,  img.shape[1] - w_width +1, stepSize):
        for y in range(0, img.shape[0] - w_height +1, stepSize):
            window.append(img[x:x + w_width, y:y + w_height])

    window = np.array(window)
    return window


#getting center_pixel for every sliding window
def get_center_pixel(window):
    x_i = int(len(window)/2)
    return window[x_i][x_i]

#getting mean of sliding window
def getMean(window):
    return window.mean()
```

```
#getting stdev of sliding window
def getStdev(window):
    return window.std()

def DVmean(arr,bw,tw):
    s = 0.0
    for i in arr:
        s += i
    return (s/((bw**2)-(tw**2)))

def DVstd(arr,bw,tw,mean):
    s = 0.0
    l = bw**2 - tw**2
    for i in range(l):
        if i > len(arr)-1:
            s += mean**2
        else:
            res = (arr[i] - mean)**2
            s += res
    return np.sqrt(s/(l-1))


#computing threshold for the sliding window
def DetectionVariable(window,arr,bw,tw):
    center_pixel = get_center_pixel(window)
    #print(center_pixel)
    win_mean = DVmean(arr,bw,tw)
    #win_mean = getMean(window)
    #print(win_mean)
    win_stdev = DVstd(arr,bw,tw,win_mean)
    #win_stdev = getStdev(window)
    #print(win_stdev)
    return ((center_pixel-win_mean)/win_stdev)

#plotting binary image for the specific threshold.
#here img_d always takes ndarray.
def shipDetection_binaryImg(img_d,threshold):
    for i in range(len(img_d)):
        if img_d[i] >= threshold:
            img_d[i] = 1      #Valid ship
        else:
            img_d[i] = 0      #not a ship
```

```
    img_size = int(math.sqrt(len(img_d)))
    img_d_img = img_d.reshape(img_size,img_size)
    return img_d_img

#Computing the target window of size 3x3 and returning along with the
#background pixels
def get_TargetWindow(img,bw,tw):
    s = []
    i = -int(tw/2)
    #print(i)
    for _ in range(tw):
        #s = [img[int(bw/2)-1][int(bw/2)-1:int(bw/2)+2], img[int(bw/2)][int(bw/2)-1:int(bw/2)+2], img
        s.append(img[int(bw/2)+i][int(bw/2)-int(tw/2):int(bw/2)+int(tw/2)+1])
        i += 1

    s = (np.array(s))

    background = []
    for z in img:
        for t in z:
            if t in s:
                pass
            else:
                background.append(t)
    background = np.array(background)
    return [s,background]

def get_GuardWindow(img,bw,gw):
    s = []
    i = -int(gw/2)
    #print(i)
    for _ in range(gw):
        #s = [img[int(bw/2)-1][int(bw/2)-1:int(bw/2)+2], img[int(bw/2)][int(bw/2)-1:int(bw/2)+2], img
        s.append(img[int(bw/2)+i][int(bw/2)-int(gw/2):int(bw/2)+int(gw/2)+1])
        i += 1

    s = (np.array(s))

    background = []
    for z in img:
        for t in z:
            if t in s:
```

```
                pass
            else:
                background.append(t)
    background = np.array(background)
    return [s,background]

def noisePower(arr,bw,tw):
    s = 0.0
    for i in arr:
        s += i
    return (s/((bw**2)-(tw**2)))

def scaleFactor(pfa,backgroundWindow_size,targetWindow_size):
    N = backgroundWindow_size**2 - targetWindow_size**2
    alpha = N*(pfa**(-1/N) -1)
    return alpha
```

In [2]:
```
m = np.array(np.arange(100))
m = m.reshape(10,10)
```

In [3]:
```
m
```

Out[3]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

In [6]:
```
guard,back = get_GuardWindow(m,10,5)
guard
```

Out[6]:
```
array([[33, 34, 35, 36, 37],
       [43, 44, 45, 46, 47],
       [53, 54, 55, 56, 57],
       [63, 64, 65, 66, 67],
       [73, 74, 75, 76, 77]])
```

In [7]:
```
back
```

Out[7]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 38,
       39, 40, 41, 42, 48, 49, 50, 51, 52, 58, 59, 60, 61, 62, 68, 69, 70,
       71, 72, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
       93, 94, 95, 96, 97, 98, 99])
```

In [8]:
```
target, back2 = get_TargetWindow(guard, 5,3)
target
```

Out[8]:
```
array([[44, 45, 46],
       [54, 55, 56],
       [64, 65, 66]])
```

In [9]:
```
back2
```

Out[9]:
```
array([33, 34, 35, 36, 37, 43, 47, 53, 57, 63, 67, 73, 74, 75, 76, 77])
```

In [ ]:
```

```