ASSIGNMENT 3

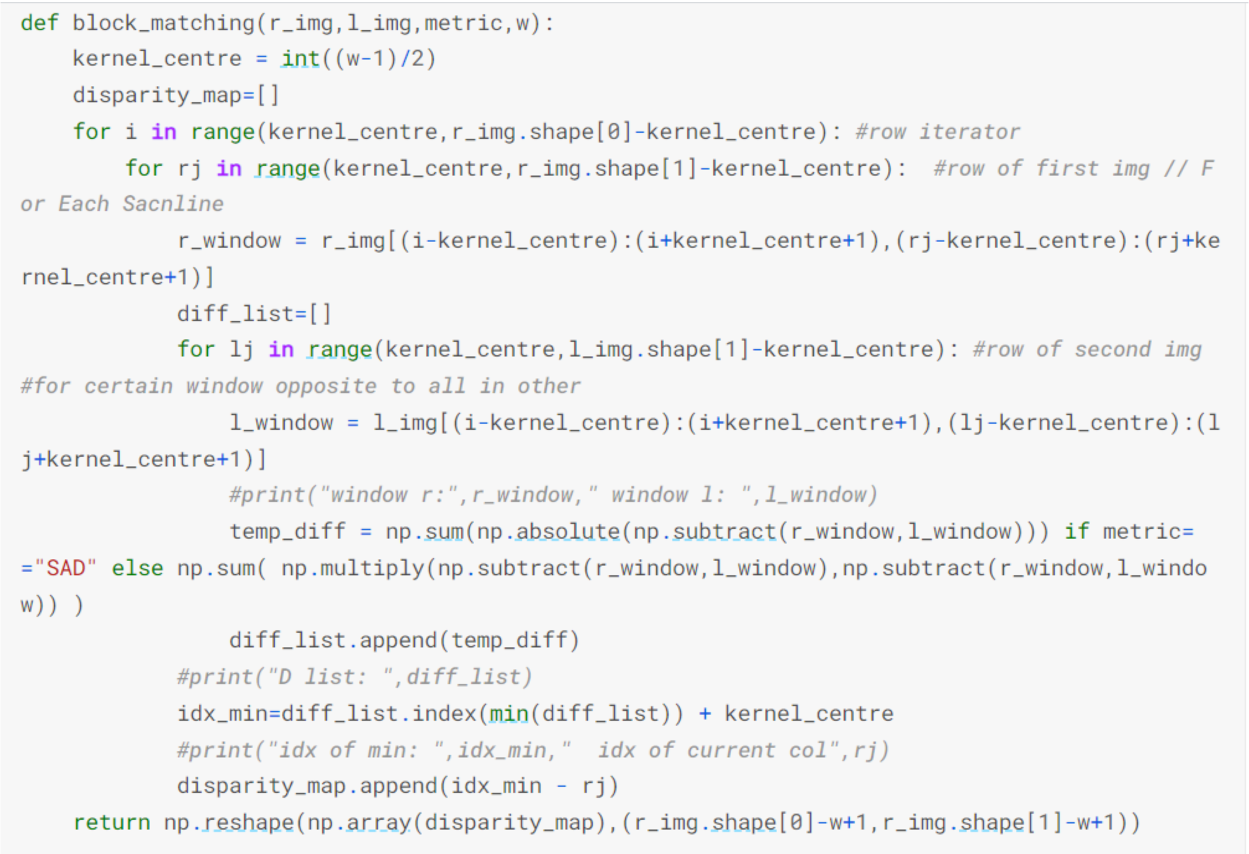
Computer vision

Ahmed Ashraf	**5803**
Abdelrahman Habib	**6019**
Osama Sherif	**6012**
Mohamed Aiman	**6017**

kaggle https://www.kaggle.com/mohammedaiman/cv-ass2-stereo-vision

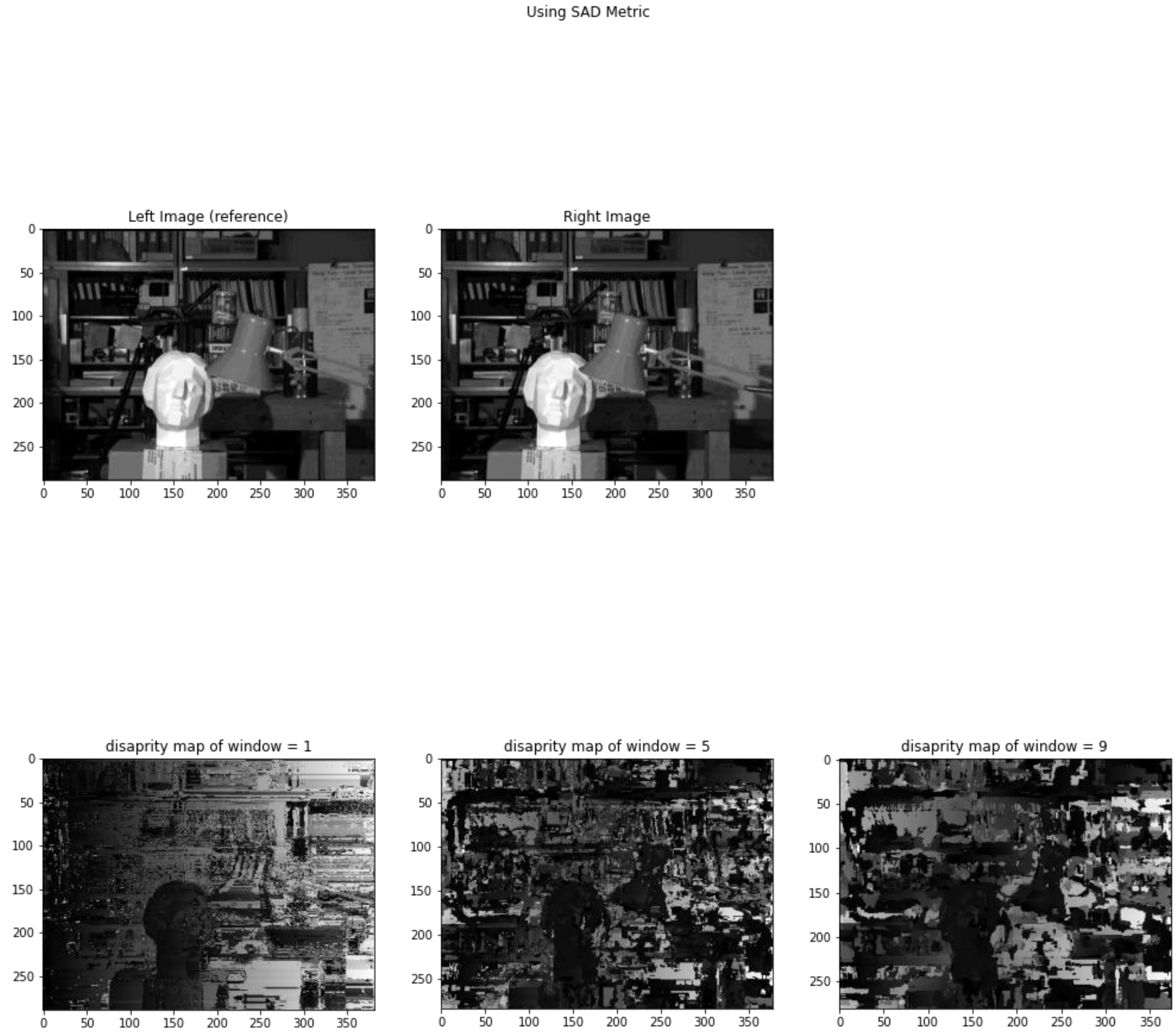
1.1 Block Matching

1.SAD & SSD block matching Function



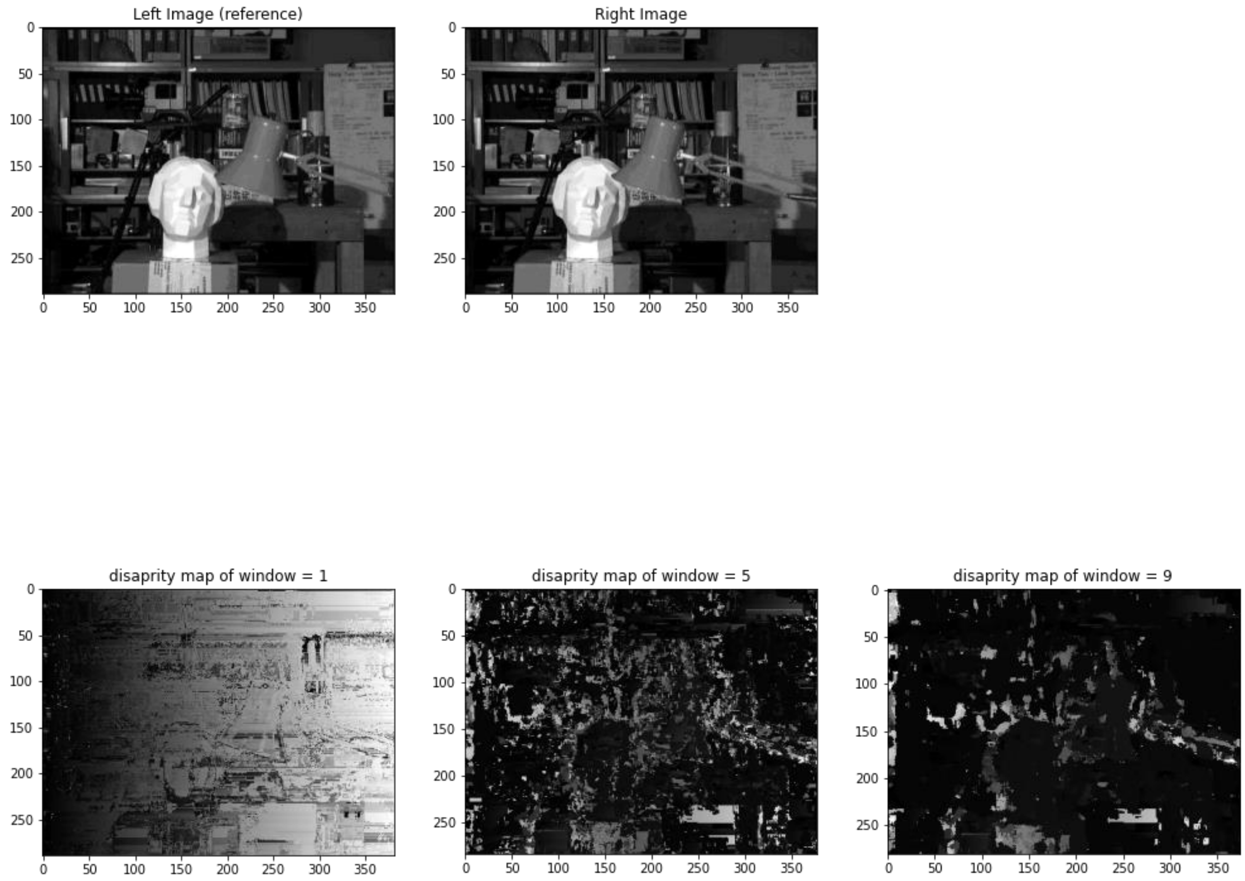
```python
def block_matching(r_img,l_img,metric,w):
    kernel_centre = int((w-1)/2)
    disparity_map=[]
    for i in range(kernel_centre,r_img.shape[0]-kernel_centre): #row iterator
        for rj in range(kernel_centre,r_img.shape[1]-kernel_centre):  #row of first img // For Each Sacnline
            r_window = r_img[(i-kernel_centre):(i+kernel_centre+1),(rj-kernel_centre):(rj+kernel_centre+1)]
            diff_list=[]
            for lj in range(kernel_centre,l_img.shape[1]-kernel_centre): #row of second img #for certain window opposite to all in other
                l_window = l_img[(i-kernel_centre):(i+kernel_centre+1),(lj-kernel_centre):(lj+kernel_centre+1)]
                #print("window r:",r_window," window l: ",l_window)
                temp_diff = np.sum(np.absolute(np.subtract(r_window,l_window))) if metric=="SAD" else np.sum( np.multiply(np.subtract(r_window,l_window),np.subtract(r_window,l_window)) )
                diff_list.append(temp_diff)
            #print("D list: ",diff_list)
            idx_min=diff_list.index(min(diff_list)) + kernel_centre
            #print("idx of min: ",idx_min,"  idx of current col",rj)
            disparity_map.append(idx_min - rj)
    return np.reshape(np.array(disparity_map),(r_img.shape[0]-w+1,r_img.shape[1]-w+1))
```

Output (SAD):


Using SAD Metric
Left Image (reference)
Right Image
disaprity map of window = 1
disaprity map of window = 5
disaprity map of window = 9
0
50
100
150
200
250
300
350

Output (SSD):

Using SSD Metric



Left Image (reference)

Right Image

disaprity map of window = 1

disaprity map of window = 5

disaprity map of window = 9

1.2 Dynamic Programming

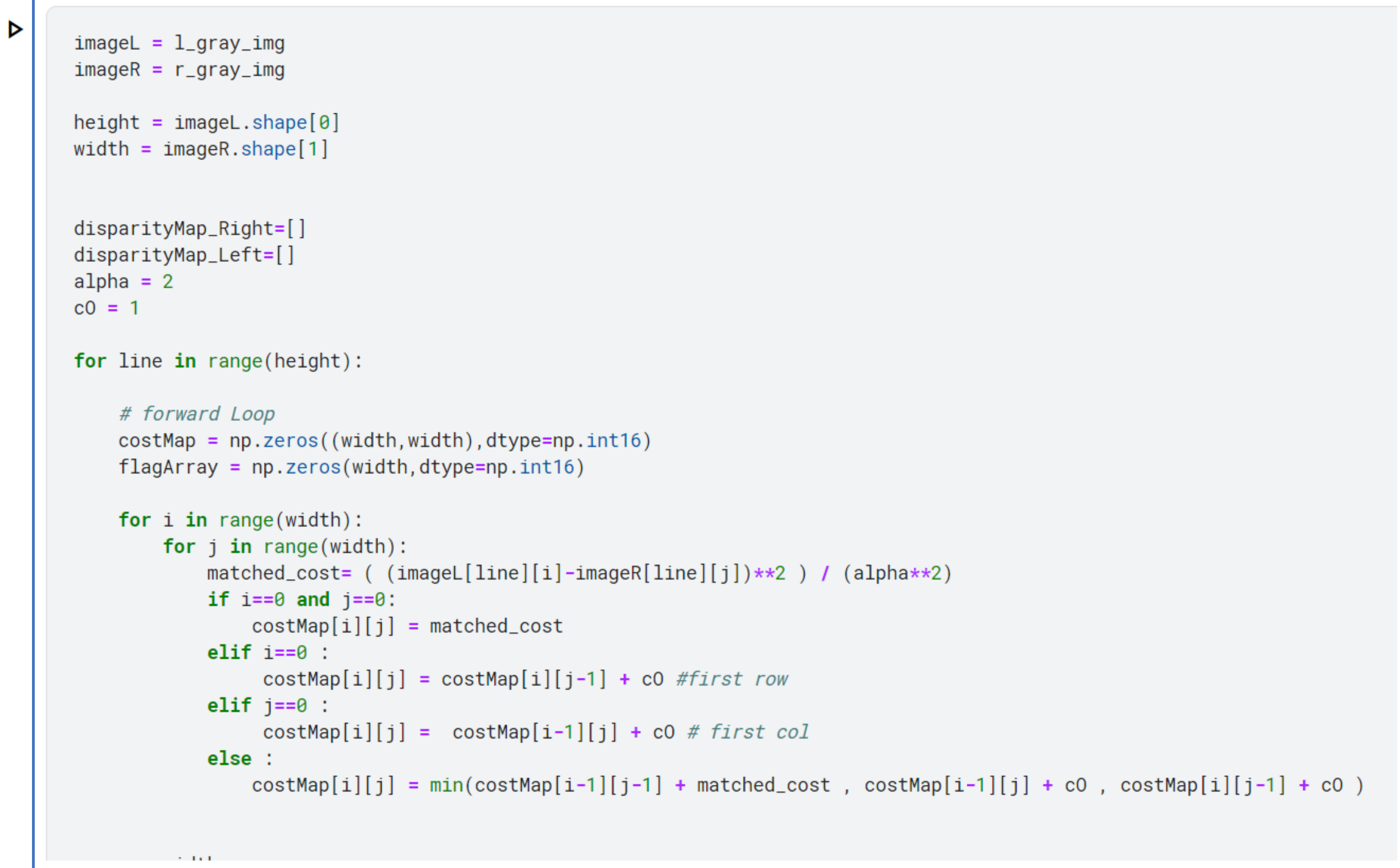
Forward Step:

Here we are filling the D Matrix on which we will perform the Backtracking Step.

We fill the D Matrix Following this two Conditions:

1. D(1, 1) = d11

2. D(i, j) = min(D(i1, j1) + dij , D(i1, j) + c0, D(i, j1) + c0)

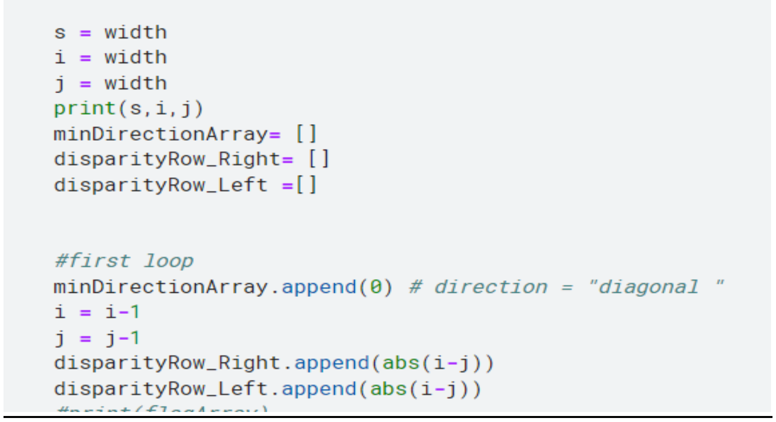


```
imageL = l_gray_img
imageR = r_gray_img

height = imageL.shape[0]
width = imageR.shape[1]


disparityMap_Right=[]
disparityMap_Left=[]
alpha = 2
c0 = 1

for line in range(height):

    # forward Loop
    costMap = np.zeros((width,width),dtype=np.int16)
    flagArray = np.zeros(width,dtype=np.int16)

    for i in range(width):
        for j in range(width):
            matched_cost= ( (imageL[line][i]-imageR[line][j])**2 ) / (alpha**2)
            if i==0 and j==0:
                costMap[i][j] = matched_cost
            elif i==0 :
                 costMap[i][j] = costMap[i][j-1] + c0 #first row
            elif j==0 :
                 costMap[i][j] =  costMap[i-1][j] + c0 # first col
            else :
                costMap[i][j] = min(costMap[i-1][j-1] + matched_cost , costMap[i-1][j] + c0 , costMap[i][j-1] + c0 )
```

Backtracking:

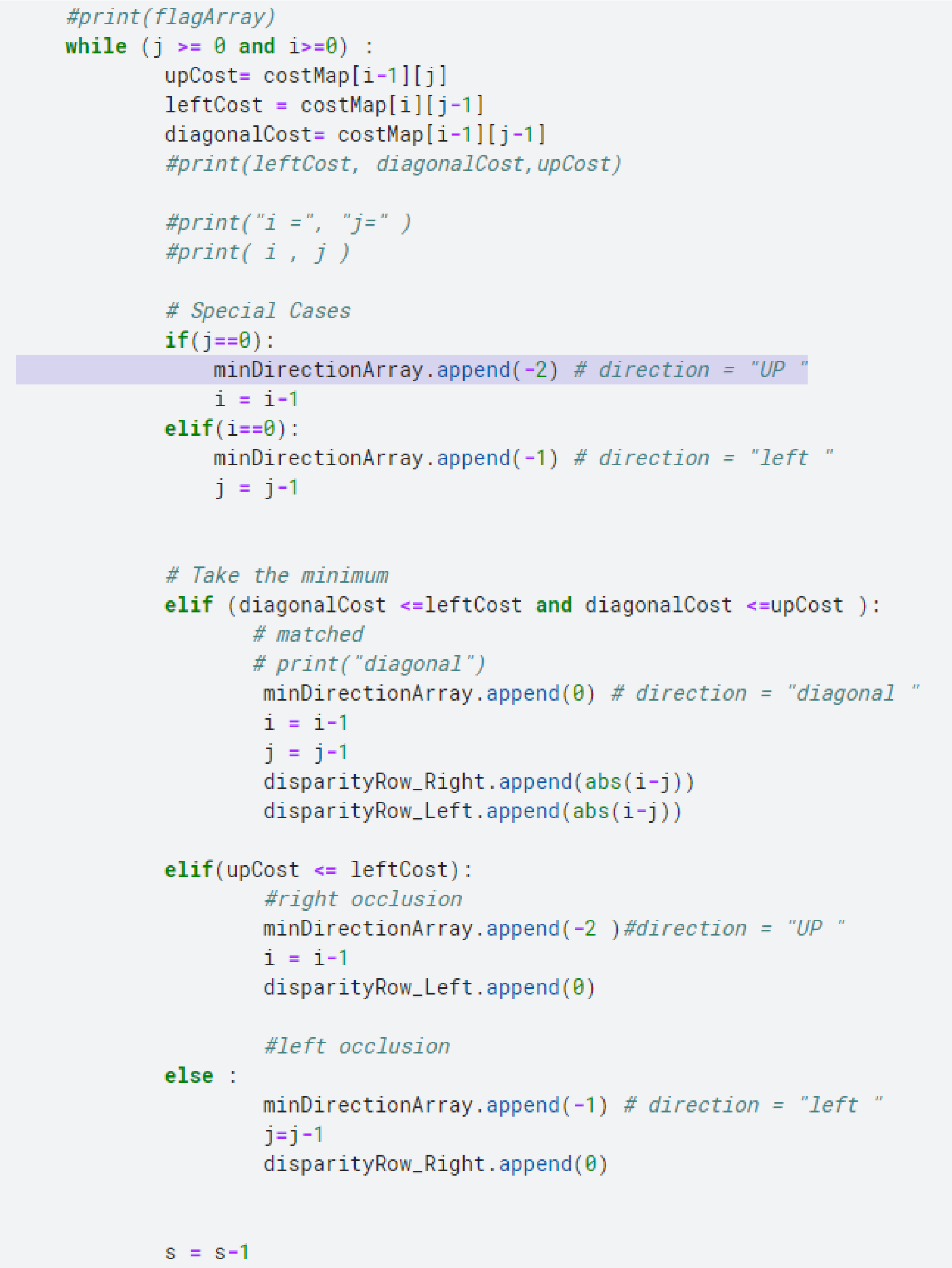
Here we Perform the Backtracking step to fill the LEFT and Right Disparity Maps:

Initializing and performing the first loop and cell D(N,N):



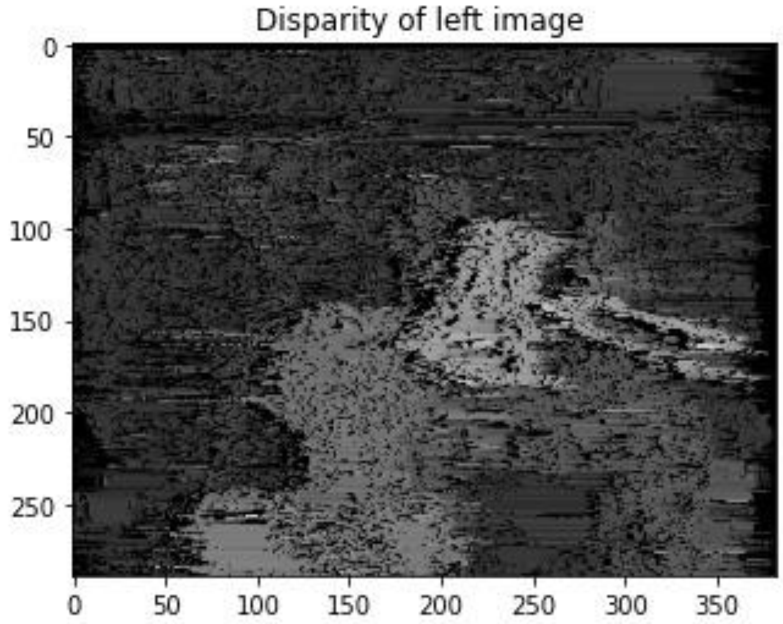
```
s = width
i = width
j = width
print(s,i,j)
minDirectionArray= []
disparityRow_Right= []
disparityRow_Left =[]


#first loop
minDirectionArray.append(0) # direction = "diagonal "
i = i-1
j = j-1
disparityRow_Right.append(abs(i-j))
disparityRow_Left.append(abs(i-j))
#print(flagArray)
```

Completing the Backtracking for the rest of D:

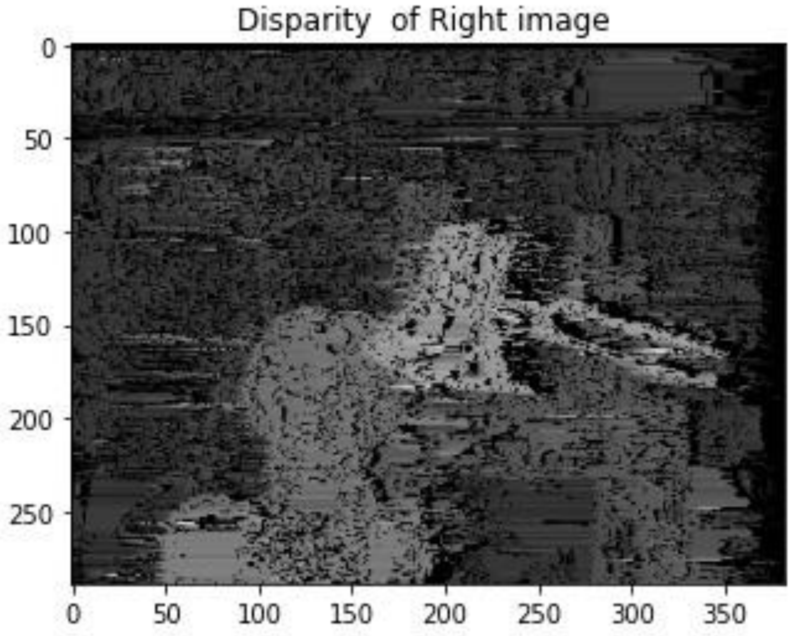


```
    #print(flagArray)
    while (j >= 0 and i>=0) :
            upCost= costMap[i-1][j]
            leftCost = costMap[i][j-1]
            diagonalCost= costMap[i-1][j-1]
            #print(leftCost, diagonalCost,upCost)

            #print("i =", "j=" )
            #print( i , j )

            # Special Cases
            if(j==0):
                minDirectionArray.append(-2) # direction = "UP "
                i = i-1
            elif(i==0):
                minDirectionArray.append(-1) # direction = "left "
                j = j-1


            # Take the minimum
            elif (diagonalCost <=leftCost and diagonalCost <=upCost ):
                   # matched
                   # print("diagonal")
                    minDirectionArray.append(0) # direction = "diagonal "
                    i = i-1
                    j = j-1
                    disparityRow_Right.append(abs(i-j))
                    disparityRow_Left.append(abs(i-j))

            elif(upCost <= leftCost):
                    #right occlusion
                    minDirectionArray.append(-2 )#direction = "UP "
                    i = i-1
                    disparityRow_Left.append(0)

                    #left occlusion
            else :
                    minDirectionArray.append(-1) # direction = "left "
                    j=j-1
                    disparityRow_Right.append(0)


            s = s-1
```

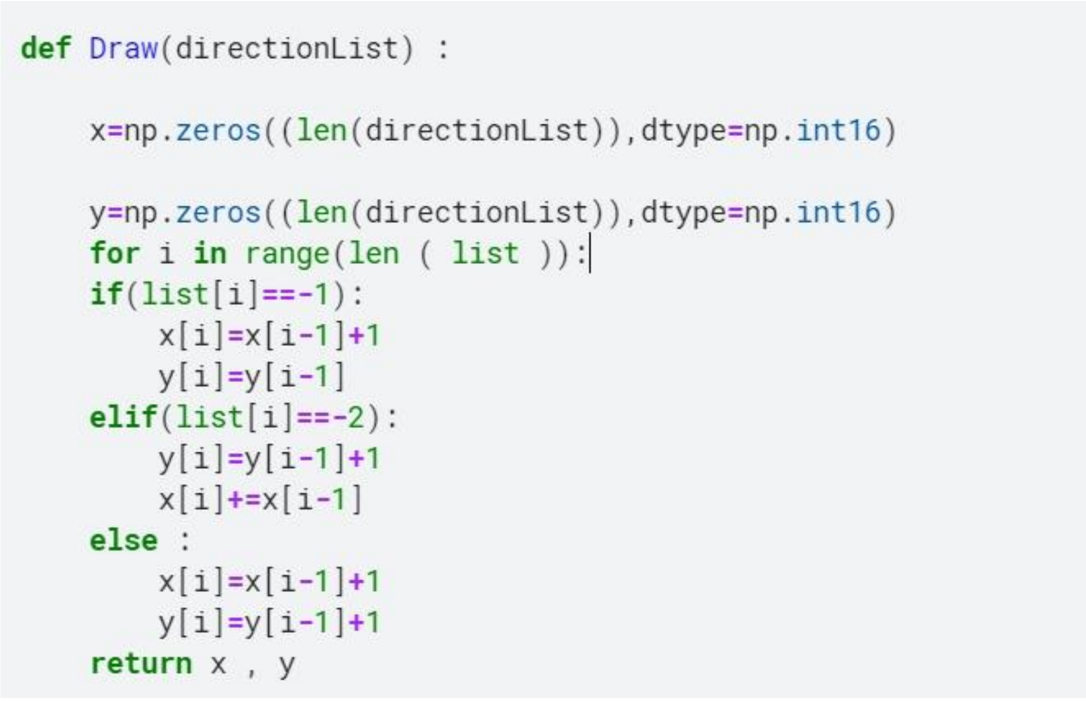
Left Disparity Map:



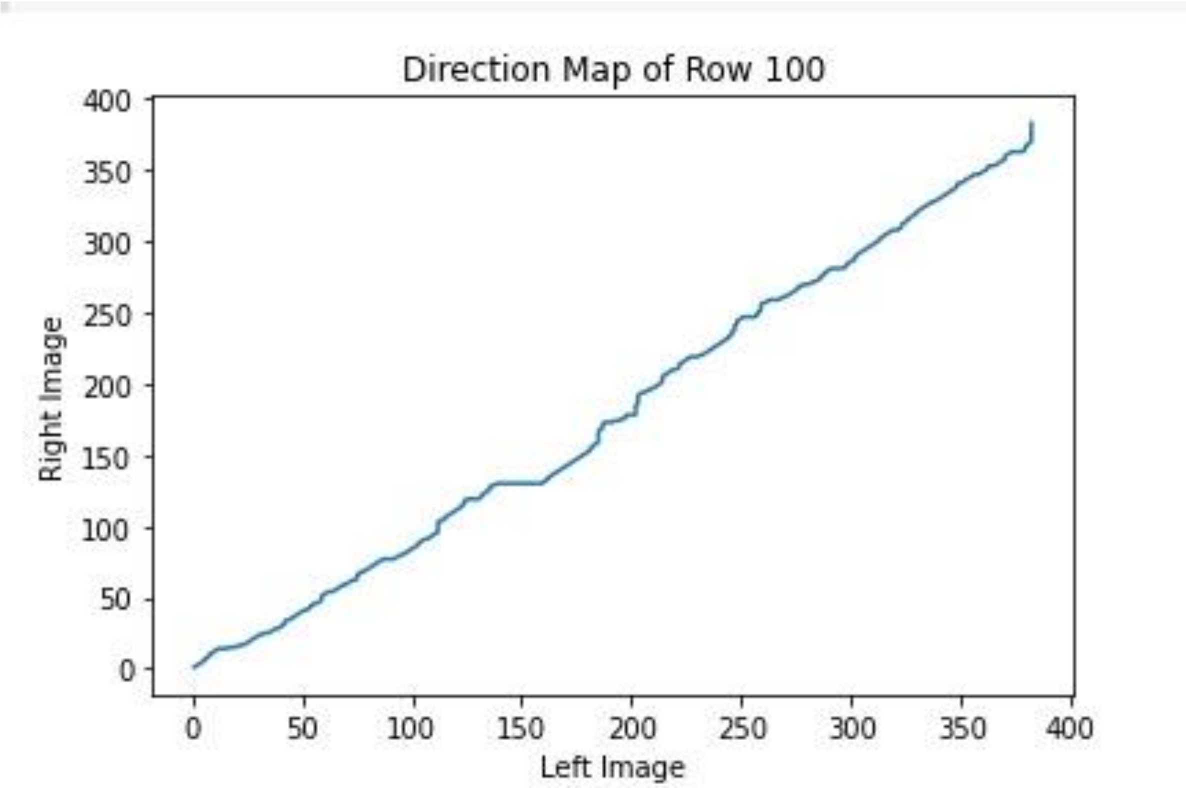
Right Disparity Map:



Finally, we represent a single row path in the Backtracking Algorithm:



```
def Draw(directionList) :

    x=np.zeros((len(directionList)),dtype=np.int16)

    y=np.zeros((len(directionList)),dtype=np.int16)
    for i in range(len ( list )):
    if(list[i]==-1):
        x[i]=x[i-1]+1
        y[i]=y[i-1]
    elif(list[i]==-2):
        y[i]=y[i-1]+1
        x[i]+=x[i-1]
    else :
        x[i]=x[i-1]+1
        y[i]=y[i-1]+1
    return x , y
```


Direction Map of Row 100
Right Image
Left Image