Assignment 2 - CS 532
Pranshu Savani

Q1 Code :

Rank Transformation:

```
def rank_transform(image, tran_windowsize) :
    w, h = image.size
    # converting image to array
    img_arr = np.asarray(image)
    # array to store transformed image
    tran_img = np.zeros((w,h), np.uint8)
    tran_img.shape = h,w
    window_half = int(tran_windowsize / 2)
    # padding black pixels around both arrays to compute complete rank transform for all pixels
    img_arr = pad_zeros(img_arr,tran_windowsize)
    tran_img = pad_zeros(tran_img,tran_windowsize)
    # rank transform
    for y in range(window_half,h+window_half):
        for x in range(window_half,w+window_half):
            rank = 0
            rank = np.sum(np.where(img_arr[y-window_half:y+window_half+1,x-window_half:x+window_half+1]>img_arr[y,x],1,0))
            tran_img[y, x] = rank
    # print(tran_img[0:5,0:5])
    # return de-padded array
    return tran_img[window_half:h+window_half,window_half:w+window_half]
```

Stereo Matching:

```
def stereo_match(left_img, right_img, save_path, sad_kernel=3, rt_kernel=5, max_disparity=64):
    left_img = Image.open(left_img).convert('L')
    right_img = Image.open(right_img).convert('L')
    w, h = left_img.size  # assuming both images are same size
    # get transformed images
    left = pad_zeros(rank_transform(left_img,rt_kernel),sad_kernel)
    right = pad_zeros(rank_transform(right_img,rt_kernel),sad_kernel)
    # initialize disparity map
    dmap = np.zeros((w, h), np.uint8)
    dmap.shape = h, w
    dmap = pad_zeros(dmap,sad_kernel)
    kernel_half = int(sad_kernel / 2)
    for y in trange(kernel_half, h + kernel_half):
        for x in range(kernel_half, w + kernel_half):
            disp = 0
            min_sad = 999999
            # loop through each candidate pixel/hypothesis
            for d in range(max_disparity):
                sad = 0
                sad_temp = 0
                # kernel size is (v,u)
                for v in range(-kernel_half, kernel_half+1):
                    for u in range(-kernel_half, kernel_half+1):
                        # sum of absolute difference in window
                        sad_temp = int(left[y+v, x+u]) - int(right[y+v, (x+u) - d])
                        sad += abs(sad_temp)
                # get least SAD and store disparity for that pixel
                if sad < min_sad:
                    min_sad = sad
                    disp = d
            # adjust range of disparity from 0 - 63 to 0 - 255 for saving image
            dmap[y, x] = disp * (255/(max_disparity-1))
    Image.fromarray(dmap[kernel_half:h+kernel_half,kernel_half:w+kernel_half]).save(save_path)
    return save_path
```

Imports and support functions:

```
import numpy as np
from PIL import Image
from tqdm import trange

def error(ground_truth,img):
    dispimg = Image.open(img).convert('L')
    ground_truth = Image.open(ground_truth).convert('L')
    d1ar = np.asarray(dispimg)
    d2ar = np.asarray(ground_truth)
    w,h = dispimg.size
    bad_pixels = 0
    for y in range(h):
        for x in range(w):
            if(abs(round(d1ar[y,x]/4) - round(d2ar[y,x]/4)) > 1):
                bad_pixels+=1
    error = bad_pixels/(w*h)
    return round(error*100,2)

def pad_zeros(arr,window_size):
    w_half = int(window_size/2)
    arr = np.pad(arr,((w_half,w_half),(w_half,w_half)),'constant')
    return arr
```

Notes:

- Padding image with black columns for getting complete rank information and stereo comparison for pixels that have kernels falling outside window.(gives for a more accurate comparison than just considering image pixels as black when window falls outside image range.)
- Testing for nearest 64 pixels on the epipolar line.(scanning entire epipolar line takes too much time, also this sets the disparities between 0-63 by default)
- Before saving, the disparity map values are adjusted to be between 0 - 255, for better image output.

Q2 Code :

```
def confidence_cost(ground, estimated, d = 12):
    MAX = 999999
    epsilon = 1E-9
    [w, h] = ground.shape
    C = np.zeros((w,h,d))
    c1, c2 =  np.ones((w,h)) * MAX, np.ones((w,h)) * MAX

    for i in range(w):
        for j in range(d,h):
            for k in range(d):
                C[i,j,k] = np.abs(ground[i,j] - estimated[i,j-d])
                if c1[i,j] == MAX:
                    c1[i,j] = C[i,j,k]
                elif c1[i,j] < C[i,j,k] and c2[i,j] == MAX:
                    c2[i,j] = C[i,j,k]
                elif c2[i,j] != MAX:
                    break
    return c1+epsilon,c2+epsilon

def confidenceanalysis(ground_, estimated_,save_path):
    ground = np.round((np.asarray(Image.open(ground_).convert('L'))/4))
    estimated = np.round((np.asarray(Image.open(estimated_).convert('L'))/4))
    #return c1 and c2 values for all pixel locations c1.shape=c2.shape=ground.shape
    c1,c2 = confidence_cost(ground, estimated)
    C_PKRN = c2 / c1
    C_flat = C_PKRN.flatten()
    mid_C_PKRN = np.sort(C_flat)[::-1][:int(len(C_flat)/2)][-1]
    mask = np.where(C_PKRN >= mid_C_PKRN, 1, 0)
    pix_kept = np.sum(mask)
    print('number of pixels kept =', pix_kept)
    bad_pix = np.sum(np.where(np.absolute(estimated - ground) > 1, 1, 0) * mask)
    err_rate = round(100 * bad_pix / pix_kept, 2)
    print('Error rate of sparse disparity map =', err_rate, '%')
    disp_mapsparse = Image.open(estimated_) * mask
    Image.fromarray(disp_mapsparse.astype(np.uint8)).save(save_path)
```

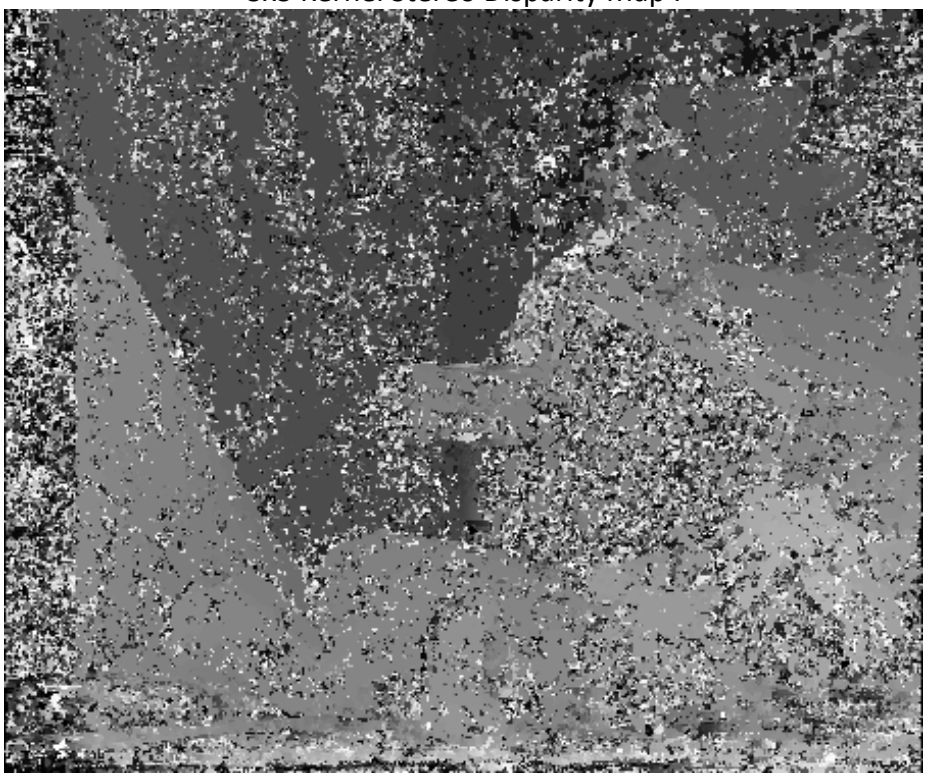
Results :

```python
if __name__ == '__main__':
    image1 = "./teddy/teddyL.pgm"
    image2 = "./teddy/teddyR.pgm"
    ground_truth = "./teddy/disp2.pgm"
    k3 = stereo_match(image1, image2, 'dmap3x3.pgm',3, 5, 64)
    print(f'error rate for 3x3 kernel stereo SAD : {error(ground_truth,k3)}')
    confidenceanalysis(ground_truth,k3,'sparse_dm3x3.pgm')
    k15 = stereo_match(image1, image2,'dmap15x15.pgm', 15, 5, 64)
    print(f'error rate for 15x15 kernel stereo SAD : {error(ground_truth,k15)}')
    confidenceanalysis(ground_truth,k15,'sparse_dm15x15.pgm')
```

✓ 14m 38.1s

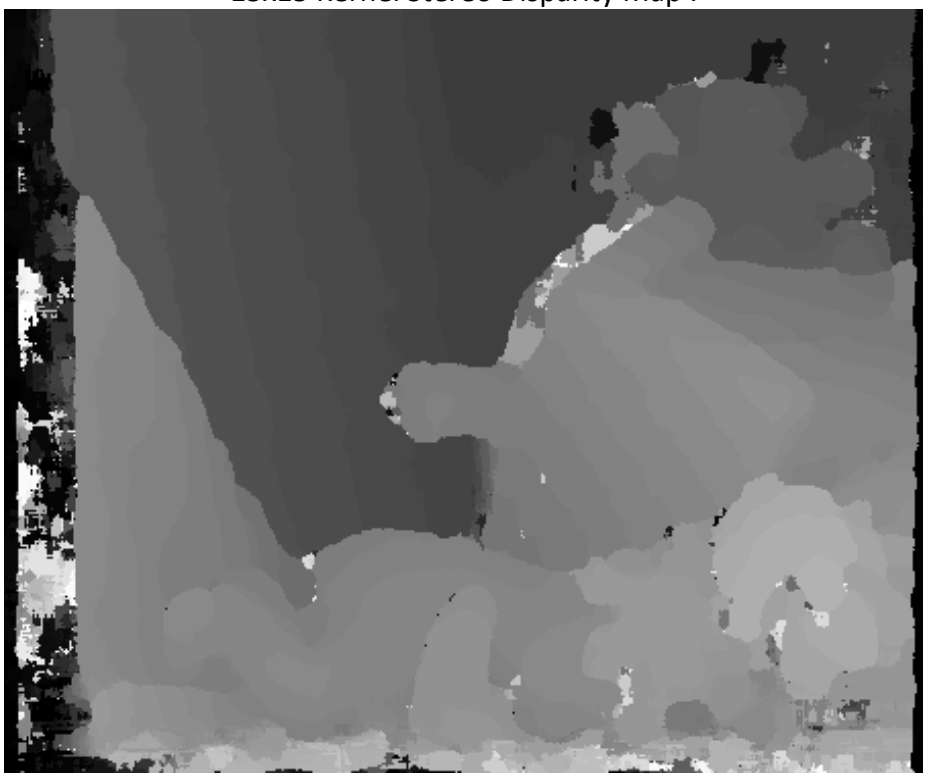
```
100%|██████████| 375/375 [00:38<00:00,  9.67it/s]

error rate for 3x3 kernel stereo SAD : 47.22
# of pix kept = 84455
Error rate of sparse disp map = 35.03 %

100%|██████████| 375/375 [13:50<00:00,  2.22s/it]

error rate for 15x15 kernel stereo SAD : 22.06
# of pix kept = 119411
Error rate of sparse disp map = 7.01 %
```

3x3 Kernel Stereo Disparity Map :



3x3 Kernel Sparse Disparity Map :

15x15 Kernel Stereo Disparity Map :



15x15 Kernel Sparse Disparity Map :