Exercise 4: Stereo Matching and DLT

Due date: 9/6/2022

In the lectures we discussed creating a depth image from two images and warping images to align them to each other. In this exercise you will implement stereo matching and build a panorama pipeline.

1 Stereo Matching

1.1 SSD

Write a function which takes two images, Left and Right, and finds outputs the disparity map using SSD.

```
def disparitySSD(img_l: np.ndarray, img_r: np.ndarray, disp_range: (int, int), k_size: int)
-> np.ndarray:
    """
    img_l: Left image
    img_r: Right image
    range: Minimum and Maximum disparity range. Ex. (10,80)
    k_size: Kernel size for computing the SSD, kernel.shape = (k_size*2+1,k_size*2+1)

    return: Disparity map, disp_map.shape = Left.shape
    """
```

SSD is defined by:

$$S_{LR}(r,c) = \sum_{i=0}^{krows} \sum_{j=0}^{kcols} [(L(i,j) - R(r+i-\frac{krows}{2}, c+j-\frac{kcols}{2})]^2$$

For each pixel(r,c), you should return the shift in x, that gives you the minimum SSD response. Try that on images pair0-L.png and pair0-R.png.

1.2 Normalized Correlation

Write a function which takes two images, Left and Right, and finds outputs the disparity map using Normalized Correlation.

```
def disparityNC(img_l: np.ndarray, img_r: np.ndarray, disp_range: (int, int), k_size: int)
-> np.ndarray:
    """
    img_l: Left image
    img_r: Right image
    range: Minimum and Maximum disparity range. Ex. (10,80)
    k_size: Kernel size for computing the SSD, kernel.shape = (k_size*2+1,k_size*2+1)

    return: Disparity map, disp_map.shape = Left.shape
    """
```

Normalized Corolation is defined by:

$$R_{LR}(r,c) = \sum_{i=0}^{krows}\sum_{j=0}^{kcols}\{L(i,j)\cdot R(r+i-\frac{krows}{2}, c+j-\frac{kcols}{2})\}$$

The result is then normalized:

$$NC_{LR}(r,c) = \frac{R_{LR}(r,c)}{\sqrt{R_{RR}(r,c)R_{LL}(\frac{krows}{2},\frac{kcols}{2})}}$$

For each pixel(r,c), you should return the shift in x, that gives you the **Maximum** response.

Try that on images pair1-L.png and pair1-R.png.

2 Homography and Warping

2.1 DLT

Write a function which takes four, or more, pairs of matching keypoints and returns the homography matrix H. Calculate the homography using DLT as shown in the presentation (Hint: SVD).

```
def computeHomography(src_pnt: np.ndarray, dst_pnt: np.ndarray) -> (np.ndarray, float):
    """
        Finds the homography matrix, M, that transforms points from src_pnt to dst_pnt.
```

```
    returns the homography and the error between the transformed points to their
    destination (matched) points.

    Error = np.sqrt(sum((M.dot(src_pnt)-dst_pnt)**2))

    src_pnt: 4+ keypoints locations (x,y) on the original image. Shape:[4+,2]
    dst_pnt: 4+ keypoints locations (x,y) on the destination image. Shape:[4+,2]

    return: (Homography matrix shape:[3,3], Homography error)
"""
```

Comments:

1. The last number in the M matrix (M[2,2]) should be 1.0!
2. Don't forget to convert the coordinates to homogeneous coordinates for calculating the error.
3. To test, try this:

input:
src_pnt = np.array([[279, 552],[372, 559],[362, 472],[277, 469]])
dst_pnt = np.array([[24, 566],[114, 552],[106, 474],[19, 481]])

output:
M =
[[1.84928168e+00 6.29286954e-01 2.00796141e+02]
[9.37137905e-02 3.12582854e+00 -5.89220129e+02]
[-5.32787017e-04 2.02578161e-03 1.00000000e+00]]
Error: 1.815947868067344

2.2 Warping

Write a function which takes two images (eg. a poster and a billboard), and warps the poster on to the billboard. The user will mark the source points and their matches on the destination image.

```
def warpImag(src_img: np.ndarray, dst_img: np.ndarray) -> None:
    """
      Displays both images, and lets the user mark 4 or more points on each image.
```

```
    Then calculates the homography and transforms the source image on to the destination image.
    Then transforms the source image onto the destination image and displays the result.

    src_img: The image that will be 'pasted' onto the destination image.
    dst_img: The image that the source image will be 'pasted' on.

    output: None.
"""
```

Comments:

1. For the clicking, use one for each window (src/dst):

```
def onclick_1(event):
    global fig1, fig2, locations_1
    print('onclick_1')
    x = event.xdata
    y = event.ydata

    plt.plot(x, y, '*r')
    plt.show()
    locations_1.append([x, y])

# display image 1
fig1 = plt.figure()
cid = fig1.canvas.mpl\_connect('button\_press\_event', onclick\_1)
plt.imshow(img1)
```

2. For the warping look at the example on 'Warping' in week: 4
3. For the stitching (pasting) use a mask.

```
mask = proj_src == 0
canvas = dst_img * mask + (1 - mask) * proj_src
```

3 Submission

Good luck!